Bài giảng:

LẬP TRÌNH WEB

Tác giả:

Họ và tên: Phạm Quang Hiếu, Đại học Kỹ thuật Công nghiệp Thái Nguyên (TNUT)

Email: quanghieu@tnut.edu.vn

Đối tượng học: Sinh viên / Người học lập trình web cơ bản đến nâng cao

Nội dung chính:

HTML, CSS, JavaScript, AJAX

JSP, JDBC

Spring MVC / Struts

Thực hành ứng dụng Web hoàn chỉnh

NĂM 2025

MỤC TIÊU MÔN HỌC

1. Về kiến thức:

- Trang bị kiến thức nền tảng về lập trình web phía Client: HTML, CSS, JavaScript, AJAX.
- Cung cấp kiến thức cơ bản về lập trình web phía Server: sử dụng JSP (JavaServer Pages) và JDBC (Java Database Connectivity) để kết nối cơ sở dữ liệu.
- Giúp sinh viên làm quen với các framework Java Web hiện đại như Struts, Spring MVC để xây dựng ứng dụng quy mô thực tế.

2. Kỹ năng:

- Làm chủ kỹ thuật lập trình web cả client-side và server-side.
- Sử dụng thành thạo thư viện, API và framework Java hỗ trợ lập trình web.
- Phát triển kỹ năng làm việc nhóm, triển khai dự án thực tế.
- Nâng cao khả năng đọc – hiểu tài liệu chuyên ngành bằng tiếng Anh.

2. Thái độ, Chuyên cần:

- Chuyên cần: tham dự đầy đủ các buổi học, thực hành và thảo luận.
- Tích cực: chủ động làm bài tập, tham gia phát biểu, đóng góp ý kiến trong lớp.
- Rèn luyện thói quen làm việc chuyên nghiệp: quản lý thời gian, tuân thủ tiến độ, hợp tác nhóm.

Contents

CHƯƠNG 1: TỔNG QUAN VỀ LẬP TRÌNH WEB

MỤC TIÊU

1. Nắm được các khái niệm liên quan đến lập trình web, các ngôn ngữ tiêu biểu cho lập trình web, các kiến thức cơ bản về HTML; Ôn tập các kiến thức về mạng máy tính và ngôn ngữ Java

2. Phân biệt các thành phần liên quan đến lập trình web phía client và phía Java; Nắm được các vấn đề đặt ra khi xây dựng ứng dụng web, cách thiết kế ứng dụng web nói chung.

NỘI DUNG CHI TIẾT

1.1 WWW và các khái niệm cơ bản

1.1.1. WWW (World Wide Web):

❖ **Khái niệm**

WWW (World Wide Web), thường được gọi là "Web", là một hệ thống thông tin toàn cầu cho phép người dùng truy cập và chia sẻ tài liệu, hình ảnh, video và các tài nguyên khác thông qua Internet.

- WWW không phải là Internet**.** Đây là điểm quan trọng nhất cần phân biệt. Internet là hạ tầng vật lý và logic gồm các mạng máy tính kết nối toàn cầu. WWW là một trong những dịch vụ lớn nhất hoạt động trên nền tảng Internet đó. Có thể hình dung Internet là hệ thống đường xá, còn WWW là các tòa nhà, cửa hàng (website) được xây dựng trên hệ thống đường đó.

- Hoạt động dựa trên mô hình client-server**.** Trình duyệt web của bạn đóng vai trò là "client" (máy khách) gửi yêu cầu, và máy chủ (server) chứa trang web sẽ phản hồi bằng cách gửi dữ liệu về cho bạn.

- Sử dụng các công nghệ tiêu chuẩn**.** WWW dựa trên ba công nghệ chính:

+ HTTP (Hypertext Transfer Protocol): Giao thức truyền tải siêu văn bản, quy định cách client và server giao tiếp với nhau.

+ HTML (Hypertext Markup Language): Ngôn ngữ đánh dấu siêu văn bản, dùng để tạo và định dạng các trang web.

+ URL (Uniform Resource Locator): Địa chỉ duy nhất của mỗi tài nguyên trên Web, giúp xác định vị trí của trang web hoặc tệp tin.

❖ **Vai trò và tầm quan trọng**

WWW đã cách mạng hóa cách chúng ta tiếp cận thông tin, học hỏi, giao tiếp và làm việc. Nó tạo ra một không gian mở, nơi mọi người có thể chia sẻ kiến thức và tương tác với nhau mà không bị giới hạn về địa lý, trở thành một phần không thể thiếu của cuộc sống hiện đại.

1.1.2. Website và Web Page

❖ **Website**

- Website (hay còn gọi là trang web tổng thể) là một tập hợp các trang web con (Web Page) được liên kết với nhau, cùng thuộc một tên miền duy nhất và được lưu trữ trên một máy chủ (web server). Website thường đại diện cho một tổ chức, doanh nghiệp, hoặc cá nhân, với mục đích cung cấp thông tin, sản phẩm, dịch vụ, hoặc giải trí.

- Hãy hình dung Website giống như một cuốn sách hoàn chỉnh. Cuốn sách này có một cái tên (tên miền) và chứa nhiều trang giấy (web page) bên trong.

Ví dụ:

+ google.com là một website. Nó bao gồm nhiều trang con như Google Search, Google Images, Google Maps, Google Drive, v.v.

+ facebook.com: Đây là một website. Các trang cá nhân, trang nhóm, trang tin tức (News Feed) đều là các trang con của website này.

+ https://www.google.com/search?q=vinasite.com.vn: Đây là một website của một công ty thiết kế web. Nó sẽ có các trang con như "Giới thiệu", "Dịch vụ", "Tin tức", "Liên hệ".

❖ **Web Page**

- Web Page (hay còn gọi là trang web con hoặc trang đơn) là một tài liệu duy nhất trên World Wide Web. Nó có một địa chỉ URL riêng biệt và là một phần của một website. Một Web Page chứa nội dung cụ thể như văn bản, hình ảnh, video, liên kết và được xây dựng bằng ngôn ngữ HTML.

- Hãy hình dung Web Page giống như một trang giấy cụ thể trong cuốn sách. Mỗi trang giấy có một nội dung riêng biệt và một số trang duy nhất.

Ví dụ:

https://vi.wikipedia.org/wiki/Website: Đây là một Web Page. Nó là một trang duy nhất nằm trong website Wikipedia.org.

https://www.google.com/doodles: Đây là một Web Page nằm trong website Google.com, chứa các hình ảnh "doodle" đặc biệt.

https://vinasite.com.vn/dich-vu-thiet-ke-website: Đây là một Web Page cụ thể trên website Vinasite, nói về dịch vụ thiết kế website.

Bảng tóm tắt sự khác biệt:

Đặc điểm	**Website**	**Web Page**
Định nghĩa	Tập hợp của nhiều Web Page được liên kết với nhau.	Một trang tài liệu duy nhất trên Web.
Quy mô	Rộng lớn, có thể bao gồm hàng trăm hoặc hàng nghìn trang.	Nhỏ hơn, là một phần của Website.
Nội dung	Chứa nhiều loại nội dung khác nhau, phục vụ nhiều mục đích.	Chứa nội dung cụ thể, thường tập trung vào một chủ đề duy nhất.
Tên gọi	Được gọi bằng tên miền (domain name) chung.	Thường được gọi bằng tên tiêu đề hoặc URL cụ thể.
Ví dụ	facebook.com	facebook.com/tên_người_dùng
	vnexpress.net	vnexpress.net/thoi-su-24h

1.1.3. Web Browser và Web Server

❖ **Web Browser**

Web Browser: (hay trình duyệt web) là một phần mềm ứng dụng được cài đặt trên máy tính, điện thoại hoặc máy tính bảng của bạn, cho phép bạn truy cập và xem các trang web trên World Wide Web.

Chức năng:

- Gửi yêu cầu: Khi bạn nhập một địa chỉ website (URL) vào thanh địa chỉ của trình duyệt và nhấn Enter, trình duyệt sẽ gửi một yêu cầu (request) đến Web Server.

- Hiển thị nội dung: Sau khi nhận được dữ liệu phản hồi từ Web Server (dưới dạng HTML, CSS, JavaScript), trình duyệt sẽ dịch và hiển thị nội dung đó thành một trang web có giao diện trực quan, dễ đọc, với đầy đủ hình ảnh, văn bản và video.

Ví dụ

- Google Chrome: Phổ biến nhất hiện nay.

- Mozilla Firefox: Nổi tiếng với sự ưu tiên quyền riêng tư.

- Microsoft Edge: Trình duyệt mặc định trên Windows.

- Safari: Trình duyệt mặc định trên các thiết bị của Apple.

❖ **Web Server**

Web Server (hay máy chủ web) là một phần mềm hoặc một máy tính chuyên dụng được dùng để lưu trữ các trang web, tệp tin và cơ sở dữ liệu. Vai trò chính của nó là xử lý các yêu cầu từ Web Browser và gửi lại các nội dung tương ứng.

Chức năng:

- Lưu trữ: Web Server lưu trữ tất cả các thành phần của một website như HTML, CSS, hình ảnh, video.

- Phản hồi: Khi nhận được yêu cầu từ một trình duyệt, Web Server sẽ tìm kiếm các tệp tin được yêu cầu và gửi chúng trở lại trình duyệt. Quá trình này được thực hiện thông qua giao thức HTTP/HTTPS.

- Xử lý yêu cầu: Với các website động (dynamic websites), Web Server còn có nhiệm vụ xử lý các yêu cầu phức tạp hơn, ví dụ như truy vấn cơ sở dữ liệu để tạo ra nội dung cá nhân hóa trước khi gửi về cho người dùng.

Ví dụ:

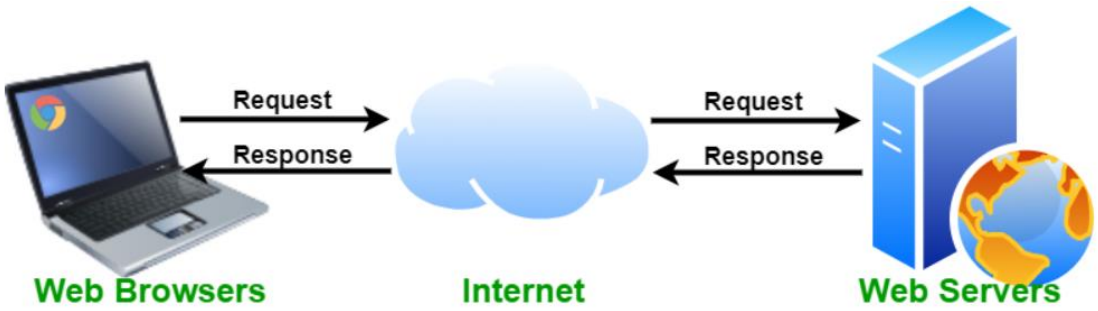
- Apache HTTP Server: Một trong những Web Server lâu đời và phổ biến nhất, mã nguồn mở.

- Microsoft Internet Information Services (IIS): Web Server của Microsoft, thường được dùng cho các trang web chạy trên nền tảng Windows.

Mối quan hệ giữa Web Browser và Web Server: Mối quan hệ giữa Web Browser và Web Server giống như mối quan hệ giữa một người khách hàng và một người phục vụ tại nhà hàng.

- Web Browser là khách hàng: Bạn ngồi tại bàn (máy tính của bạn) và yêu cầu một món ăn (một trang web).

- Web Server là người phục vụ: Người phục vụ sẽ nhận yêu cầu, đi vào bếp (nơi lưu trữ website), lấy món ăn (các tệp tin của trang web), và mang ra phục vụ bạn (gửi dữ liệu về trình duyệt để hiển thị).



1.1.4. HTTP/HTTPS:

❖ **HTTP**

HTTP (Hypertext Transfer Protocol) là giao thức nền tảng của World Wide Web, được dùng để truyền tải thông tin giữa Web Server và Web Browser. Nó giống như một bộ quy tắc cho phép trình duyệt của bạn (client) và máy chủ (server) "nói chuyện" với nhau.

Cách hoạt động:

- Khi bạn gõ một địa chỉ website vào trình duyệt, trình duyệt sẽ tạo một yêu cầu HTTP (HTTP request) và gửi đến máy chủ.

- Máy chủ nhận yêu cầu, xử lý và gửi lại phản hồi HTTP (HTTP response) chứa các tài nguyên của trang web (như file HTML, hình ảnh, CSS...).

- Cuối cùng, trình duyệt nhận phản hồi và hiển thị trang web cho bạn.

Đặc điểm nổi bật: Không bảo mật; Dữ liệu được truyền tải dưới dạng văn bản thuần túy, không được mã hóa. Điều này có nghĩa là bất kỳ ai có thể theo dõi đường truyền đều có thể đọc được thông tin của bạn, chẳng hạn như mật khẩu hoặc thông tin thẻ tín dụng.

❖ **HTTPS**

HTTPS (Hypertext Transfer Protocol Secure) là phiên bản bảo mật của HTTP. Nó sử dụng một công nghệ mã hóa là SSL/TLS (Secure Sockets Layer/Transport Layer Security) để mã hóa toàn bộ dữ liệu được truyền giữa trình duyệt và máy chủ.

Cách hoạt động:

- Khi bạn truy cập một trang web HTTPS, trình duyệt và máy chủ sẽ thực hiện một "bắt tay" (handshake) để xác thực và thiết lập một kết nối an toàn.

- Dữ liệu sau đó sẽ được mã hóa trước khi gửi đi và chỉ được giải mã bởi máy chủ hoặc trình duyệt.

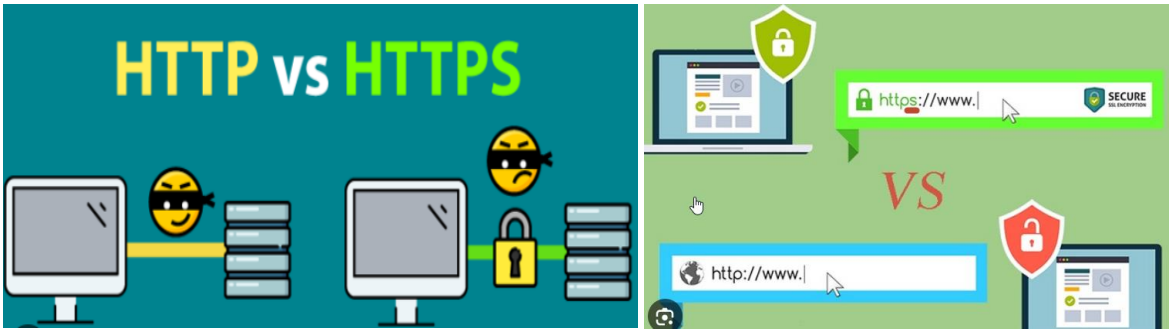
Đặc điểm nổi bật:

- Bảo mật: Dữ liệu được mã hóa, giúp bảo vệ thông tin cá nhân của người dùng khỏi những kẻ tấn công (hacker).

- Được khuyến khích sử dụng: Các trình duyệt hiện đại thường hiển thị biểu tượng ổ khóa trên thanh địa chỉ để báo hiệu rằng trang web đó đang sử dụng HTTPS. Các công cụ tìm kiếm như Google cũng ưu tiên các trang web sử dụng HTTPS vì tính bảo mật cao.

So sánh HTTP và HTTPS:

Đặc điểm	HTTP	HTTPS
Mục đích	Truyền tải dữ liệu trên web	Truyền tải dữ liệu an toàn trên web
Cổng mặc định	Cổng 80	Cổng 443
Bảo mật	Không mã hóa, dễ bị tấn công	Mã hóa, cực kỳ an toàn
Yêu cầu	Không cần chứng chỉ SSL/TLS	Bắt buộc có chứng chỉ SSL/TLS
Nhận diện	Địa chỉ bắt đầu bằng http://	Địa chỉ bắt đầu bằng https:// và có biểu tượng ổ khóa



1.1.5. URL (Uniform Resource Locator):

- URL (Uniform Resource Locator) là một chuỗi ký tự dùng để xác định vị trí của một tài nguyên trên Internet. Về cơ bản, nó chính là "địa chỉ web" mà bạn thường thấy trên thanh địa chỉ của trình duyệt. Mỗi tài nguyên trên Web, dù là một trang web, hình ảnh, video hay tệp tin, đều có một URL duy nhất.

- Hãy hình dung URL giống như địa chỉ nhà của một người. Nó cho phép bạn biết chính xác nơi để tìm và truy cập ngôi nhà đó.

- Cấu trúc URL: [giao thức]://[tên miền]/[đường dẫn]

- Ví dụ: https://moet.gov.vn/van-ban

+ **https**: Giao thức.

+ moet.gov.vn: Tên miền.

+ **/van-ban:** Đường dẫn đến tài nguyên cụ thể.

1.1.6. DNS (Domain Name System)

DNS (Domain Name System), hay Hệ thống Tên miền, là một hệ thống phân giải tên miền thành địa chỉ IP. Nó giống như một "danh bạ điện thoại" của Internet, giúp chuyển đổi tên miền dễ nhớ của con người (ví dụ: google.com) thành địa chỉ IP dạng số mà máy tính có thể hiểu được (ví dụ: 142.250.199.14).

Máy tính giao tiếp với nhau bằng các địa chỉ IP (dãy số phức tạp). Tuy nhiên, con người rất khó để nhớ hàng trăm địa chỉ IP khác nhau. DNS ra đời để giải quyết vấn đề này, giúp chúng ta chỉ cần nhớ tên miền.

- Dễ nhớ: Thay vì phải gõ 142.250.199.14 để truy cập Google, bạn chỉ cần gõ google.com.

- Linh hoạt: Nếu Google thay đổi địa chỉ IP của họ, hệ thống DNS sẽ được cập nhật. Bạn vẫn có thể truy cập google.com mà không cần biết địa chỉ IP mới.

1.2 Khái quát về lập trình web

1.2.1. Khái niệm

Lập trình web là việc sử dụng các ngôn ngữ lập trình và công nghệ để xây dựng và duy trì các trang web (websites) và ứng dụng web (web applications) hoạt động trên World Wide Web. Nó bao gồm cả việc thiết kế giao diện (cái mà người dùng nhìn thấy) và xây dựng hệ thống xử lý logic phía sau.

Lập trình web được chia thành hai mảng chính:

- Lập trình Front-end (phía máy khách): Xây dựng giao diện người dùng, nơi họ tương tác trực tiếp. Công việc này đảm bảo trang web thân thiện, đẹp mắt và hoạt động trơn tru trên mọi thiết bị.

- Lập trình Back-end (phía máy chủ): Xây dựng hệ thống "hậu trường," nơi xử lý dữ liệu, kết nối với cơ sở dữ liệu và quản lý các logic nghiệp vụ.

1.2.2. Công nghệ cốt lõi

Đây là ba ngôn ngữ và công nghệ nền tảng để lập trình web:

- **HTML (Hypertext Markup Language):** Ngôn ngữ đánh dấu để tạo cấu trúc và nội dung cho trang web. Nó giống như bộ xương của một trang web, định hình các thành phần như tiêu đề, đoạn văn, hình ảnh và liên kết.

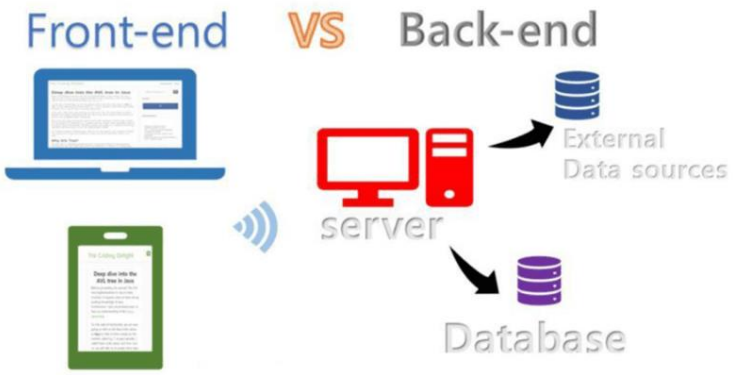
- **CSS (Cascading Style Sheets):** Ngôn ngữ dùng để định dạng và trang trí giao diện. CSS giúp trang web trở nên đẹp mắt, có màu sắc, bố cục và hiệu ứng phù hợp. Nó là lớp da và quần áo của trang web.

- **JavaScript:** Ngôn ngữ lập trình giúp tạo ra các tương tác động trên trang web. Ví dụ: các hiệu ứng chuyển động, xử lý dữ liệu từ form, hoặc thay đổi nội dung mà không cần tải lại trang. JavaScript mang lại sự sống và sự tương tác cho trang web.

1.2.3. Phân biết lập trình Front-end và Back-end

- **Front-end**: Là những gì người dùng nhìn thấy và tương tác. Một lập trình viên Front-end tập trung vào trải nghiệm người dùng, sử dụng HTML, CSS, JavaScript cùng với các thư viện và framework như React, Angular hoặc Vue.js.

- **Back-end**: Là hệ thống xử lý phía sau, mà người dùng không thể nhìn thấy trực tiếp. Một lập trình viên Back-end tập trung vào logic, bảo mật và cơ sở dữ liệu. Họ sử dụng các ngôn ngữ như Node.js, Python, Java, PHP và các hệ quản trị cơ sở dữ liệu như MySQL, PostgreSQL, MongoDB.



1.2.4. Mô hình MVC trong lập trình web

MVC (Model – View – Controller) là một mẫu thiết kế phần mềm, được sử dụng rộng rãi trong lập trình web để tổ chức mã nguồn. Mục đích chính của MVC là tách biệt các thành phần của ứng dụng thành ba phần độc lập, giúp việc quản lý, phát triển và bảo trì trở nên dễ dàng hơn.

Ta có thể hình dung mô hình MVC giống như quy trình phục vụ tại một nhà hàng:

- **Model** (Bộ xử lý dữ liệu): Giống như nhà bếp của nhà hàng. Đây là nơi xử lý tất cả các logic nghiệp vụ và dữ liệu của ứng dụng. Model không quan tâm đến cách dữ liệu được hiển thị, nó chỉ lo việc lưu trữ, truy vấn và cập nhật dữ liệu.

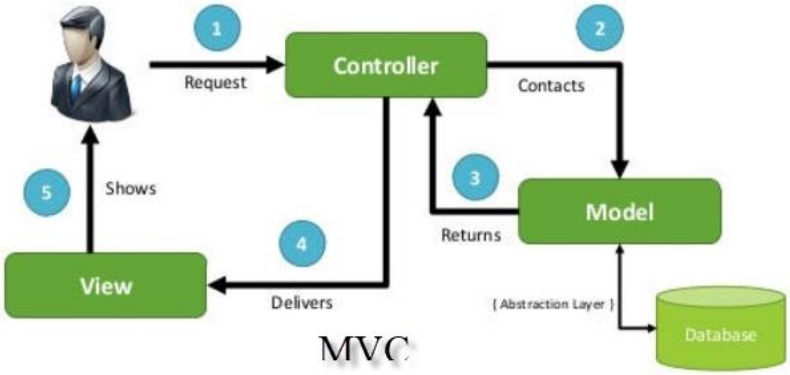
Ví dụ: Khi bạn muốn xem danh sách sản phẩm, Model sẽ truy vấn cơ sở dữ liệu để lấy danh sách sản phẩm đó.

- **View** (Giao diện người dùng): Giống như bàn ăn và thực đơn của nhà hàng. Đây là nơi hiển thị thông tin cho người dùng và tiếp nhận các tương tác từ họ. View chỉ có nhiệm vụ trình bày dữ liệu, không có logic xử lý phức tạp.

Ví dụ: Trang web hiển thị danh sách sản phẩm dưới dạng một bảng hoặc một danh sách.

- **Controller** (Bộ điều khiển): Giống như người phục vụ của nhà hàng. Controller đóng vai trò là cầu nối trung gian, nhận yêu cầu từ người dùng, gọi Model để xử lý dữ liệu và sau đó yêu cầu View hiển thị kết quả. Controller không làm gì với dữ liệu, nó chỉ điều phối.

Ví dụ: Khi người dùng nhấn vào nút "Thêm sản phẩm", Controller sẽ nhận yêu cầu này, gọi Model để thực hiện việc thêm sản phẩm vào cơ sở dữ liệu, sau đó yêu cầu View hiển thị thông báo thành công.



1.2.5. Công cụ lập trình web

❖ **Công cụ phát triển Front-end**

Dùng để xây dựng giao diện, trải nghiệm người dùng

Nhóm công cụ	Ví dụ / Ý nghĩa
Trình soạn thảo / IDE	VS Code, Sublime Text, Atom (nhẹ, nhanh cho HTML/CSS/JS)
Ngôn ngữ – Công nghệ cốt lõi	HTML, CSS, JavaScript.
Framework / Thư viện	React, Angular, Vue.js, Bootstrap, TailwindCSS.
Công cụ thiết kế & nguyên mẫu	Figma, Adobe XD.
Kiểm thử & debug trên trình duyệt	Chrome DevTools, Firefox DevTools.
Package managers	npm, Yarn (quản lý thư viện JS).

❖ **Công cụ phát triển Back-end**

Dùng để xử lý dữ liệu, logic nghiệp vụ, giao tiếp với CSDL

Nhóm công cụ	Ví dụ / Ý nghĩa
IDE / Môi trường phát triển	NetBeans (Java Web), IntelliJ IDEA, Eclipse; Visual Studio (C#).
Máy chủ ứng dụng / Web Server	Apache Tomcat (Servlet/JSP), GlassFish (Java EE), XAMPP/WAMP (PHP), Node.js server.
Framework / Công nghệ back-end	Java EE/Spring, .NET, Django, Laravel, Express.js.
CSDL & Công cụ quản trị	MySQL Workbench, SQL Server Management Studio, pgAdmin (PostgreSQL).
Công cụ kiểm thử API	Postman, SoapUI, Insomnia.
Quản lý mã nguồn	Git, GitHub, GitLab, Bitbucket.
Công cụ ảo hóa/triển khai	Docker, Kubernetes (mức giới thiệu).

1.2.6. Bảo mật cơ bản trong lập trình web

Bảo mật web là tập hợp các biện pháp kỹ thuật và quy trình nhằm bảo vệ ứng dụng, dữ liệu và người dùng khỏi các truy cập trái phép hoặc tấn công. Khi lập trình web, ngay cả ở mức cơ bản, lập trình viên cần hiểu "an toàn là thiết kế" chứ không phải "vá lỗi sau này".

❖ ***HTTPS (Hypertext Transfer Protocol Secure)***

- Là phiên bản bảo mật của HTTP, sử dụng **SSL/TLS** để mã hóa dữ liệu truyền giữa trình duyệt và máy chủ.

- Giúp ngăn chặn nghe lén, đánh cắp thông tin đăng nhập, dữ liệu cá nhân.

- Đối với sinh viên: biết cấu hình HTTPS trên web server

❖ ***Xác thực (Authentication)***

- Quá trình xác minh danh tính người dùng (đăng nhập bằng tài khoản, OTP…).

- Các cơ chế phổ biến: username/password, token, OAuth, SSO.

- Cần lưu ý hash mật khẩu trước khi lưu vào DB (không lưu plain text).

❖ ***Phân quyền (Authorization)***

- Xác định người dùng đã xác thực được phép làm gì: xem, thêm, sửa, xóa dữ liệu…
- Thường dựa trên role (quyền admin, user…) hoặc policy.
- Giúp tránh lạm quyền hoặc truy cập tài nguyên trái phép.

❖ ***Phòng chống SQL Injection***

- SQL Injection là tấn công chèn câu lệnh SQL độc hại qua input người dùng.
- Cách phòng tránh:

+ Luôn dùng prepared statements / parameterized queries thay vì nối chuỗi SQL.
+ Kiểm tra, lọc dữ liệu đầu vào.
+ Hạn chế quyền truy cập DB.

Prepared Statements (câu lệnh chuẩn bị trước): Đây là các câu lệnh SQL được biên dịch và "chuẩn bị" một lần trên máy chủ cơ sở dữ liệu. Sau đó bạn chỉ việc truyền các giá trị tham số vào để thực thi nhiều lần. Điều này giúp tăng hiệu năng và giảm lỗi bảo mật.

Parameterized Queries (truy vấn có tham số): Là các truy vấn SQL mà bạn không ghép chuỗi trực tiếp giá trị từ người dùng vào SQL, mà thay bằng các "tham số" (dấu ?). Sau đó bạn dùng hàm setString, setInt… để truyền giá trị vào. Điều này ngăn ngừa SQL Injection vì dữ liệu được xử lý an toàn tách biệt với câu lệnh SQL.

Ví dụ:

Cách cũ không an toàn:

```
String masv = request.getParameter("masv");
String sql = "SELECT * FROM SinhVien WHERE masv = '" + masv + "'";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

Ở đây nếu người dùng nhập B24DTCN071' OR '1'='1 thì toàn bộ dữ liệu có thể bị lộ.

Cách đúng với PreparedStatement / Parameterized Query:

```
String masv = request.getParameter("masv"); // dữ liệu từ user
// Sử dụng ? làm tham số, không ghép chuỗi trực tiếp
String sql = "SELECT * FROM SinhVien WHERE masv = ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, masv); // truyền giá trị vào dấu ? đầu tiên
ResultSet rs = ps.executeQuery();
while (rs.next()) {
    System.out.println("Mã SV: " + rs.getString("masv"));
    System.out.println("Họ đệm: " + rs.getString("hodem"));
    System.out.println("Tên: " + rs.getString("ten"));
    System.out.println("Lớp: " + rs.getString("lop"));
}
```

Giải thích:

`?` chính là chỗ để truyền giá trị vào (parameterized query).

`ps.setString(1, masv);` gán giá trị cho tham số thứ nhất.

SQL đã được "chuẩn bị trước" nên không bị chèn lệnh độc hại.

Ưu điểm:

An toàn hơn: chống SQL Injection.

Nhanh hơn: câu lệnh SQL được chuẩn bị một lần, chạy nhiều lần.

Code gọn gàng: dễ bảo trì.

❖ ***Phòng chống XSS (Cross-Site Scripting)***

- XSS xảy ra khi hacker chèn mã JavaScript độc hại vào trang web để chạy trên trình duyệt người dùng.

- Cách phòng tránh:

+ Escape/encode dữ liệu đầu ra (output escaping).

+ Hạn chế cho phép nhập HTML tùy ý.

+ Sử dụng Content Security Policy (CSP) khi triển khai.

1.2.7. Quy trình hoạt động của một trang web

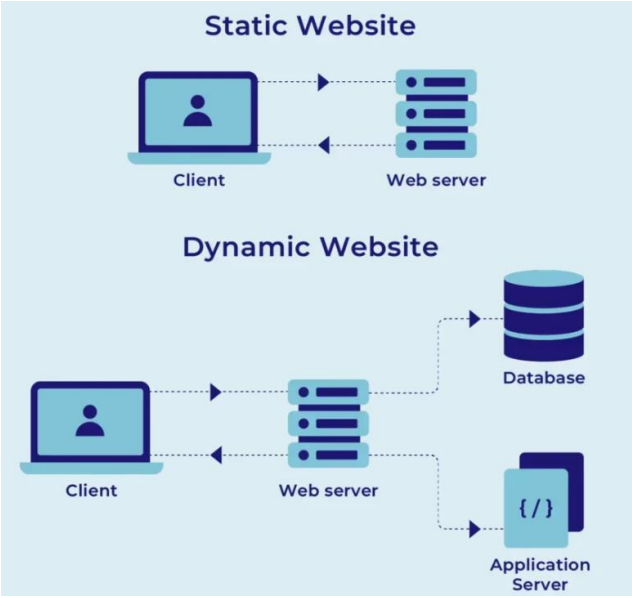
Ta có thể tóm tắt quy trình này để học viên hiểu được cách các công nghệ tương tác với nhau:

- Người dùng nhập địa chỉ web (URL) vào trình duyệt.

- Trình duyệt web (Web Browser) gửi yêu cầu đến máy chủ web (Web Server).

- Máy chủ web xử lý yêu cầu, có thể truy vấn cơ sở dữ liệu (Database), sau đó trả về dữ liệu cần thiết.

- Trình duyệt nhận dữ liệu (HTML, CSS, JavaScript), biên dịch và hiển thị trang web trên màn hình của người dùng.



1.2.8. Quy trình thiết kế và lập trình ứng dụng web

Quy trình phát triển một ứng dụng web không chỉ đơn thuần là viết code, mà là một chuỗi các bước có tổ chức, đảm bảo sản phẩm cuối cùng đáp ứng được yêu cầu và hoạt động hiệu quả. Dưới đây là các giai đoạn chính của quy trình này.

1. Giai đoạn lập kế hoạch

Xác định yêu cầu và mục tiêu: Đây là bước khởi đầu. Bạn cần tìm hiểu kỹ nhu cầu của khách hàng hoặc người dùng.

- Thu thập yêu cầu về chức năng (Ví dụ: chức năng đăng nhập, giỏ hàng, tìm kiếm).

- Xác định đối tượng người dùng (Ví dụ: học sinh, doanh nhân, người dùng phổ thông).

- Phác thảo phạm vi và các mục tiêu chính của hệ thống.

2. Giai đoạn thiết kế

- Thiết kế giao diện và trải nghiệm người dùng (UI/UX): Giai đoạn này tập trung vào việc tạo ra giao diện trực quan và dễ sử dụng.

+ Vẽ wireframe: Phác thảo khung sườn cơ bản của các trang.

+ Tạo mockup: Thiết kế chi tiết giao diện với màu sắc, phông chữ và hình ảnh.

+ Vẽ sơ đồ luồng tương tác: Xác định đường đi của người dùng qua các trang khác nhau.

- Thiết kế kiến trúc và cơ sở dữ liệu: Đây là bước thiết kế hệ thống "hậu trường".

+ Mô hình hóa dữ liệu: Xác định các bảng dữ liệu, mối quan hệ giữa chúng và các khóa chính, khóa ngoại.

+ Thiết kế kiến trúc: Lên kế hoạch về cách các thành phần Front-end và Back-end sẽ tương tác với nhau (ví dụ: mô hình MVC).

3. Giai đoạn phát triển

- Phát triển Front-end: Lập trình phần giao diện để người dùng có thể tương tác.

+ Sử dụng HTML, CSS, JavaScript để xây dựng cấu trúc, định dạng và hành vi của các trang.

+ Tận dụng các framework/thư viện như React, Angular hoặc Vue.js để tăng tốc độ phát triển.

- Phát triển Back-end: Xây dựng "bộ não" của ứng dụng.

+ Xây dựng các API để Front-end có thể gửi và nhận dữ liệu.

+ Lập trình các logic nghiệp vụ (ví dụ: xử lý đơn hàng, tính toán điểm số).

+ Kết nối với cơ sở dữ liệu để lưu trữ và truy xuất thông tin.

4. Giai đoạn hoàn thiện

- Kiểm thử và gỡ lỗi: Một ứng dụng không thể hoàn chỉnh nếu không trải qua quá trình này.

+ Kiểm tra chức năng: Đảm bảo các tính năng hoạt động đúng như yêu cầu.

+ Kiểm tra giao diện: Đảm bảo giao diện hiển thị chính xác trên nhiều trình duyệt và thiết bị khác nhau.

+ Kiểm tra bảo mật và hiệu năng: Tìm kiếm lỗ hổng bảo mật và tối ưu tốc độ tải trang.

- Triển khai lên máy chủ: Đưa ứng dụng từ môi trường phát triển lên môi trường thực tế để người dùng có thể truy cập.

+ Cấu hình Web Server (như Apache, Nginx) hoặc Application Server (như Tomcat).

+ Đảm bảo ứng dụng chạy ổn định trên máy chủ.

Bảo trì và nâng cấp: Công việc không dừng lại sau khi triển khai.

+ Theo dõi hoạt động, khắc phục các lỗi phát sinh.

+ Cập nhật tính năng mới và vá các lỗ hổng bảo mật định kỳ.

Việc tuân thủ quy trình này sẽ giúp xây dựng một ứng dụng web chuyên nghiệp, dễ quản lý và có khả năng mở rộng trong tương lai.

CHƯƠNG 2: LẬP TRÌNH CLIENT VỚI HTML, CSS VÀ JAVASCRIPT

MỤC TIÊU

1. Nắm được vai trò của CSS và các cách thức sử dụng CSS vào trang HTML; Nắm được các thành phần cơ bản của CSS; Nắm được cách viết JavaScript.

2. Biết sử dụng CSS để thay đổi định dạng trang web; Biết cách sử dụng Java Script để tương tác với người dùng qua Form.

3. Áp dụng linh hoạt CSS và Java Script để tương tác với người dùng phía Client.

NỘI DUNG CHI TIẾT

2.1. Ngôn ngữ HTML

2.1.1. Giới thiệu về HTML và vai trò

❖ HTML là gì?

Là chữ viết tắt của Hypertext Markup Language (Ngôn ngữ đánh dấu siêu văn bản). Nó không phải là một ngôn ngữ lập trình, mà là một ngôn ngữ đánh dấu. HTML sử dụng các thẻ (tags) để tạo và định dạng cấu trúc của một trang web.

❖ Vai trò của HTML

Vai trò chính của HTML là xây dựng cấu trúc và nội dung cho trang web. Dưới đây là các chức năng cụ thể:

- Tạo nội dung: HTML cho phép bạn chèn các loại nội dung khác nhau như văn bản, hình ảnh, video, âm thanh vào trang web.

- Định dạng văn bản: Bạn có thể dùng HTML để tạo tiêu đề (headings), đoạn văn (paragraphs), danh sách (lists) và nhấn mạnh văn bản (in đậm, in nghiêng).

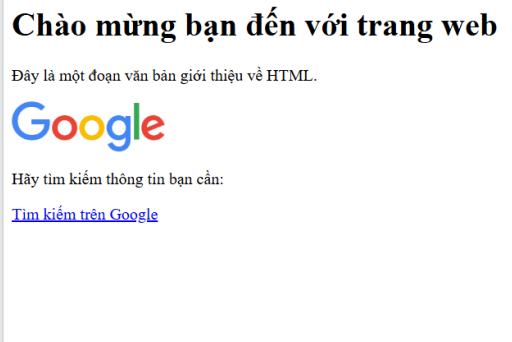
- Thiết lập các liên kết: HTML sử dụng thẻ `<a>` để tạo các siêu liên kết (hyperlinks), cho phép người dùng di chuyển từ trang này sang trang khác. Đây là yếu tố cốt lõi tạo nên "Web".

- Tạo cấu trúc biểu mẫu: HTML cho phép bạn xây dựng các biểu mẫu (forms) để thu thập thông tin từ người dùng, ví dụ như form đăng nhập, đăng ký hay liên hệ.

- Nhúng đa phương tiện: HTML có thể nhúng các nội dung đa phương tiện như video từ YouTube hoặc bản đồ từ Google Maps.

Ví dụ đoạn code HTML và kết quả hiển thị trên trình duyệt Web

```html
<html>
<head>
    <title>Trang web của tôi</title>
</head>
<body>

    <h1>Chào mừng bạn đến với trang web</h1>
    <p>Đây là một đoạn văn bản giới thiệu về HTML.</p>

    <img src="https://www.google.com/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png"
         alt="Logo Google"
         width="150">

    <p>Hãy tìm kiếm thông tin bạn cần:</p>
    <a href="https://www.google.com">Tìm kiếm trên Google</a>

</body>
</html>
```



2.1.2. Cấu trúc cơ bản của một trang HTML (doctype, html, head, body)

Mỗi tài liệu HTML đều có một cấu trúc chuẩn và bắt buộc, được tạo thành từ các thẻ (tags) lồng vào nhau. Việc hiểu rõ cấu trúc này là nền tảng để bạn có thể xây dựng bất kỳ trang web nào. Dưới đây là các thành phần chính:

<!DOCTYPE html>

Đây là dòng đầu tiên của mọi tài liệu HTML. Nó không phải là một thẻ HTML, mà là một khai báo cho trình duyệt biết rằng đây là một tài liệu HTML phiên bản 5 (HTML5) - phiên bản chuẩn và phổ biến nhất hiện nay. Khai báo này giúp trình duyệt hiển thị trang web một cách chính xác.

<HTML>...</HTML>

Thẻ này là phần tử gốc (root element) của toàn bộ trang. Nó bao bọc tất cả các nội dung còn lại. Mọi thứ bên trong thẻ `<html>` đều là một phần của tài liệu HTML. Thẻ này thường đi kèm với thuộc tính `lang` để khai báo ngôn ngữ của trang web, ví dụ `<html lang="vi">` cho tiếng Việt.

<HEAD>...</HEAD>

Phần `head` chứa các thông tin meta về trang web, tức là những thông tin không hiển thị trực tiếp trên trình duyệt mà người dùng nhìn thấy. Các thông tin này rất quan trọng đối với trình duyệt và các công cụ tìm kiếm (như Google).

- Tiêu đề trang (`<title>`): Đặt tên cho trang web, hiển thị trên thanh tab của trình duyệt.

- Tập ký tự (`<meta charset="UTF-8">`): Khai báo bảng mã ký tự để trình duyệt hiển thị các ký tự đặc biệt (như tiếng Việt có dấu) một cách chính xác.

- Liên kết với CSS, JavaScript: Chứa các đường dẫn tới các tệp tin CSS và JavaScript để định dạng và thêm tương tác cho trang web.

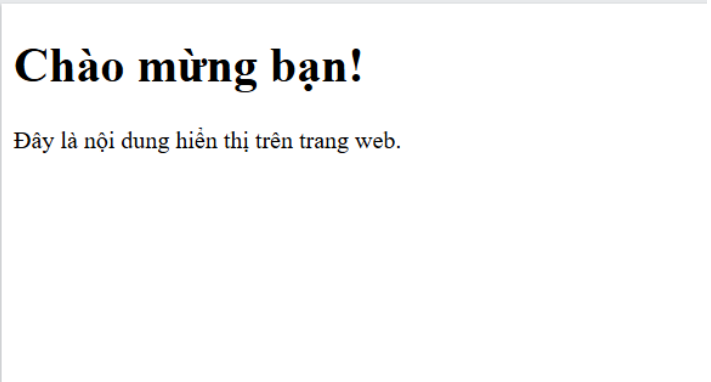
<BODY>...</BODY>

- Đây là phần quan trọng nhất, chứa toàn bộ nội dung hiển thị trên trang web. Mọi thứ mà người dùng nhìn thấy và tương tác đều nằm trong thẻ `<body>`, bao gồm văn bản, hình ảnh, liên kết, bảng, danh sách và biểu mẫu.

- `<body>` giống như phần thân của trang web, nơi chứa mọi nội dung thực tế.

Ví dụ: Dưới đây là một ví dụ minh họa đầy đủ cấu trúc của một trang HTML cơ bản:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Trang web đầu tiên của tôi</title>
</head>
<body>

    <h1>Chào mừng bạn!</h1>
    <p>Đây là nội dung hiển thị trên trang web.</p>

</body>
</html>
```



2.1.3. Các thẻ HTML thường dùng (heading, paragraph, link, image, list, table, form…)

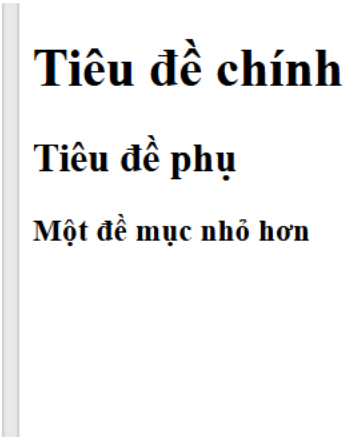
1. Thẻ tiêu đề (Heading)

- Thẻ: `<h1>` đến `<h6>`

- Ý nghĩa: Dùng để tạo các tiêu đề hoặc đề mục. `<h1>` là tiêu đề quan trọng nhất (cấp 1), và `<h6>` là tiêu đề ít quan trọng nhất (cấp 6).

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>

  <h1>Tiêu đề chính</h1>
  <h2>Tiêu đề phụ</h2>
  <h3>Một đề mục nhỏ hơn</h3>

</body>
</html>
```



Sử dụng thẻ heading đúng cách không chỉ giúp trang web có cấu trúc rõ ràng mà còn rất quan trọng cho SEO (tối ưu hóa công cụ tìm kiếm)

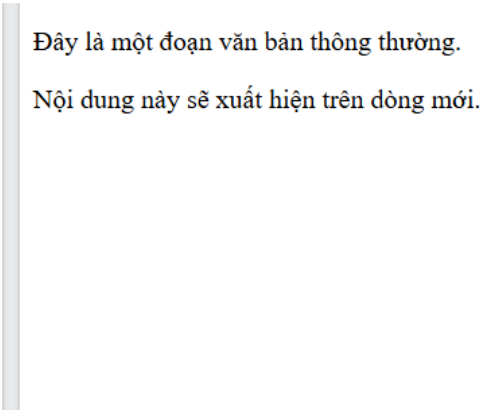
2. Thẻ đoạn văn (Paragraph)

- Thẻ: `<p>`

- Ý nghĩa: Dùng để định nghĩa một đoạn văn bản. Mỗi thẻ `<p>` sẽ tự động tạo một khoảng trống phía trên và dưới, giúp văn bản dễ đọc hơn.

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>

<p>Đây là một đoạn văn bản thông thường.</p>
<p>Nội dung này sẽ xuất hiện trên dòng mới.</p>

</body>
</html>
```



3. Thẻ liên kết (Link)

- Thẻ: `<a>` (anchor)

- Ý nghĩa: Dùng để tạo siêu liên kết, cho phép người dùng chuyển từ trang này sang trang khác hoặc truy cập các tài nguyên bên ngoài. Thuộc tính quan trọng nhất là `href` (hypertext reference), chỉ định địa chỉ đích.

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
<a href="https://www.google.com">Tìm kiếm trên Google</a>
</body>
</html>
```

Tìm kiếm trên Google

4. Thẻ hình ảnh (Image)

- Thẻ: `<img>`

- Ý nghĩa: Chèn hình ảnh vào trang web. Đây là một thẻ tự đóng (không có thẻ kết thúc).

- Các thuộc tính chính:

`src` (source): Chỉ định đường dẫn đến file hình ảnh.

`alt` (alternative text): Cung cấp văn bản thay thế, quan trọng cho SEO và khả năng tiếp cận.

`width`, `height`: Xác định kích thước của hình ảnh.

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
    <img
src="https://www.google.com/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png"
         alt="Logo Google"
         width="150">
</body>
</html>
```

Google

5. Thẻ danh sách (List)

- Danh sách không có thứ tự:

+ Thẻ: `<ul>` (unordered list) và `<li>` (list item)

+ Ý nghĩa: Tạo danh sách các mục không cần sắp xếp theo thứ tự. Các mục sẽ được hiển thị bằng các dấu chấm.

+ Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <ul>
      <li>HTML</li>
      <li>CSS</li>
      <li>JavaScript</li>
  </ul>
</body>
</html>
```

- HTML
- CSS
- JavaScript

- Danh sách có thứ tự:

+ Thẻ: `<ol>` (ordered list) và `<li>` (list item)

+ Ý nghĩa: Tạo danh sách các mục có thứ tự. Các mục sẽ được đánh số.

+ Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <ol>
      <li>Bước 1: Tìm hiểu HTML</li>
      <li>Bước 2: Nắm vững CSS</li>
      <li>Bước 3: Học JavaScript</li>
  </ol>
</body>
</html>
```

1. Bước 1: Tìm hiểu HTML
2. Bước 2: Nắm vững CSS
3. Bước 3: Học JavaScript

6. Thẻ bảng (Table)

- Thẻ: `<table>` (table), `<tr>` (table row), `<th>` (table header), `<td>` (table data)

- Ý nghĩa: Dùng để hiển thị dữ liệu dưới dạng bảng.

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <table>
    <tr>
      <th>Sản phẩm</th>
      <th>Giá</th>
    </tr>
    <tr>
      <td>Laptop</td>
      <td>15,000,000 VNĐ</td>
    </tr>
  </table>
</body>
</html>
```

Sản phẩm	Giá
Laptop	15,000,000 VNĐ

7. Thẻ biểu mẫu (Form)

- Thẻ: `<form>`, `<input>`, `<label>`, `<button>`

- Ý nghĩa: Dùng để tạo các biểu mẫu cho phép người dùng nhập dữ liệu (như form đăng nhập, đăng ký, liên hệ).

- Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <form action="/submit-form" method="post">
    <label for="name">Họ và tên:</label><br>
    <input type="text" id="name" name="name"><br>
    <button type="submit">Gửi đi</button>
  </form>
</body>
</html>
```

2.1.4. Thuộc tính (Attributes) và cách sử dụng

- **Thuộc tính (Attributes)** là những thông tin bổ sung được thêm vào các thẻ HTML để cung cấp thêm chi tiết hoặc chức năng cho một phần tử. Thuộc tính luôn được đặt trong thẻ mở đầu và thường bao gồm hai phần: **tên thuộc tính** và **giá trị thuộc tính**, được viết dưới dạng `name="value"`.

- Bạn có thể hình dung: nếu một thẻ HTML (`<img>`) là một phần tử cơ bản, thì các thuộc tính (`src`, `alt`, `width`) chính là những đặc điểm cụ thể (ví dụ: nguồn ảnh, mô tả, kích thước) giúp phần tử đó trở nên đầy đủ và hữu ích hơn.

❖ **Cấu trúc cơ bản: Thuộc tính luôn được đặt trong thẻ mở đầu của phần tử.**

```
<tên_thẻ tên_thuộc_tính="giá_trị"> Nội dung </tên_thẻ>
```

- Ví dụ:

```
<a href="https://vnexpress.net" target="_blank">Đọc báo VnExpress</a>
```

Trong ví dụ trên:

+ `href` là tên thuộc tính, có giá trị là `"https://vnexpress.net"`. Nó chỉ định địa chỉ liên kết.

+ `target` là tên thuộc tính, có giá trị là `"_blank"`. Nó chỉ định rằng liên kết sẽ được mở trong một tab mới.

❖ **Các thuộc tính phổ biến**

Dưới đây là một số thuộc tính phổ biến và quan trọng mà sinh viên cần nắm vững:

`<id và class>`:

- **`id`**: Dùng để gán một **định danh duy nhất** cho một phần tử. Mỗi `id` chỉ được sử dụng một lần trên một trang HTML.

- **`class`**: Dùng để gán một hoặc nhiều **lớp** cho một phần tử. Một `class` có thể được sử dụng cho nhiều phần tử khác nhau. Cả `id` và `class` đều rất quan trọng khi sử dụng CSS để định dạng hoặc JavaScript để thao tác với các phần tử.

`<style>`:

Dùng để **viết CSS trực tiếp** vào một phần tử. Thường được sử dụng cho các định dạng nhỏ, không phải là cách tốt nhất cho các dự án lớn.

<src và **alt>**:

src: Viết tắt của source. Dùng cho các thẻ như <img>, <script>, <audio>, <video> để chỉ định **đường dẫn** đến nguồn tài nguyên.

alt: Viết tắt của alternative text. Dùng cho thẻ <img> để cung cấp **văn bản thay thế** khi hình ảnh không tải được. Điều này giúp tăng khả năng tiếp cận (accessibility).

<href và target>:

- **href**: Viết tắt của hypertext reference. Dùng cho thẻ <a> để chỉ định **địa chỉ liên kết**.

- **target**: Dùng cho thẻ <a> để xác định nơi mở liên kết (_blank để mở tab mới).

<type và name>:

- Dùng cho các thẻ biểu mẫu như <input> để xác định loại đầu vào (ví dụ: text, email, password, submit) và tên của trường dữ liệu.

Việc sử dụng thuộc tính một cách hiệu quả giúp các phần tử HTML trở nên linh hoạt và có ý nghĩa hơn, cho phép chúng ta tạo ra các trang web phong phú về cả nội dung lẫn chức năng.

2.1.5. Form và các thành phần nhập liệu (input, select, textarea, button)

Form (biểu mẫu) là một phần tử quan trọng trong HTML, cho phép người dùng nhập thông tin và gửi chúng đến máy chủ web. Form là cầu nối chính giữa người dùng và ứng dụng web của bạn, được sử dụng trong các trường hợp như đăng ký tài khoản, đăng nhập, liên hệ, hoặc đặt hàng.

1. Thẻ <form>

- Vai trò: Định nghĩa một biểu mẫu HTML. Đây là thẻ bao bọc toàn bộ các thành phần nhập liệu khác.

- Các thuộc tính quan trọng:

+ **action**: Xác định URL mà dữ liệu biểu mẫu sẽ được gửi đến khi người dùng nhấn nút "gửi".

+ **method**: Xác định phương thức HTTP để gửi dữ liệu (GET hoặc POST). Phương thức POST thường được dùng để gửi dữ liệu nhạy cảm như mật khẩu vì nó an toàn hơn.

2. Các thành phần nhập liệu

Các thành phần này được đặt bên trong thẻ <form> để thu thập thông tin từ người dùng.

Thẻ <input>

Đây là thành phần nhập liệu phổ biến nhất, với thuộc tính type để xác định loại dữ liệu đầu vào.

type="text": Trường nhập liệu văn bản một dòng.

Ví dụ: <input type="text" name="username">

type="password": Trường nhập liệu mật khẩu, các ký tự sẽ bị ẩn đi.

Ví dụ: <input type="password" name="password">

type="email": Trường nhập liệu email. Trình duyệt sẽ tự động kiểm tra định dạng email hợp lệ.

Ví dụ: `<input type="email" name="email">`

`type="radio"`: Tạo một nút chọn duy nhất trong một nhóm. Chỉ một nút có thể được chọn tại một thời điểm.

Ví dụ: `<input type="radio" name="gender" value="male"> Nam`

`type="checkbox"`: Tạo một hộp kiểm, cho phép người dùng chọn nhiều tùy chọn.

Ví dụ: `<input type="checkbox" name="hobbies" value="reading"> Đọc sách`

`type="submit"`: Tạo một nút để gửi dữ liệu biểu mẫu.

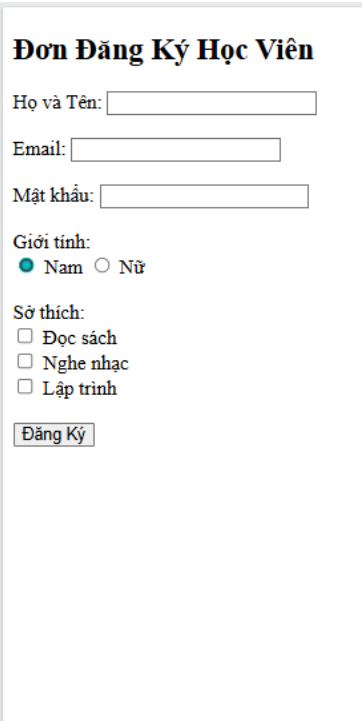
Ví dụ: `<input type="submit" value="Gửi đi">`

`type="file"`: Tạo một trường cho phép người dùng tải tệp tin lên.

Ví dụ: `<input type="file" name="avatar">`

Ví dụ thể:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Ví dụ Về Form</title>
</head>
<body>
    <h2>Đơn Đăng Ký Học Viên</h2>
    <form action="/submit-registration" method="post">
        <label for="full_name">Họ và Tên:</label>
        <input type="text" id="full_name" name="full_name" required><br><br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email_address" required><br><br>
        <label for="password">Mật khẩu:</label>
        <input type="password" id="password" name="user_password" required><br><br>
        <label>Giới tính:</label><br>
        <input type="radio" id="male" name="gender" value="Nam" checked>
        <label for="male">Nam</label>
        <input type="radio" id="female" name="gender" value="Nữ">
        <label for="female">Nữ</label><br><br>
        <label>Sở thích:</label><br>
        <input type="checkbox" id="reading" name="hobbies" value="Đọc sách">
        <label for="reading">Đọc sách</label><br>
        <input type="checkbox" id="music" name="hobbies" value="Nghe nhạc">
        <label for="music">Nghe nhạc</label><br>
        <input type="checkbox" id="coding" name="hobbies" value="Lập trình">
        <label for="coding">Lập trình</label><br><br>
        <input type="submit" value="Đăng Ký">
    </form>
</body>
</html>
```

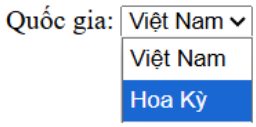


Thẻ <select> và <option>

- Vai trò: Tạo một danh sách thả xuống (dropdown list) để người dùng chọn một tùy chọn. Thẻ `<select>` bao bọc các thẻ `<option>`.

- Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <label for="country">Quốc gia:</label>
  <select name="country" id="country">
    <option value="vn">Việt Nam</option>
    <option value="us">Hoa Kỳ</option>
  </select>
</body>
</html>
```

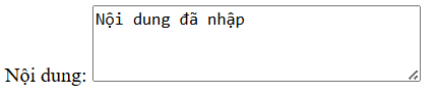


Thẻ <textarea>

- Vai trò: Tạo một trường nhập liệu văn bản nhiều dòng, thường dùng cho các trường "Nội dung" hoặc "Ghi chú".

- Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
  <label for="message">Nội dung:</label>
  <textarea name="message" id="message" rows="4" cols="50"> </textarea>
</body>
</html>
```



Thẻ <button>

- Vai trò: Tạo một nút bấm. Mặc định, nút này có `type="submit"` và sẽ gửi dữ liệu form.
- Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Tìm hiểu về các thẻ HTML</title>
</head>
<body>
    <button type="submit">Đăng ký</button>
</body>
</html>
```

2.1.6. HTML5 và các thẻ mới (audio, video, semantic tags…)

❖ HTML5 là gì?

HTML5 là phiên bản mới nhất của ngôn ngữ đánh dấu siêu văn bản HTML. Nó không chỉ là một bản cập nhật nhỏ mà là một cuộc cách mạng trong phát triển web, mang đến những tính năng mạnh mẽ để xây dựng các ứng dụng web phức tạp và tương tác. HTML5 giúp làm giảm sự phụ thuộc vào các công nghệ cũ như Adobe Flash.

❖ Các thẻ HTML5 mới quan trọng

Thẻ đa phương tiện (Media Tags)

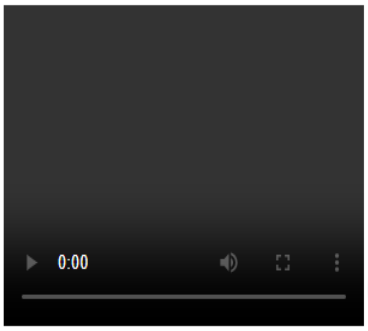
Trước đây, để nhúng video và âm thanh, lập trình viên thường phải dùng các plugin của bên thứ ba (như Flash). HTML5 đã giải quyết vấn đề này bằng cách giới thiệu các thẻ riêng biệt.

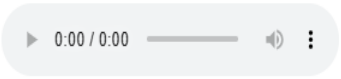
- **<audio>**: Dùng để nhúng và phát file âm thanh trực tiếp trên trình duyệt. Bạn có thể thêm các thuộc tính như `controls` để hiển thị nút phát, dừng, `autoplay` để tự động phát, và `loop` để lặp lại.

- **<video>**: Dùng để nhúng và phát video. Nó cũng có các thuộc tính tương tự như `<audio>` (như `controls`, `autoplay`).

Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Thẻ HTML5</title>
</head>
<body>
  <video width="320" height="240" controls>
    <source src="video.mp4" type="video/mp4">
    Trình duyệt của bạn không hỗ trợ thẻ video.
  </video>
  <audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    Trình duyệt của bạn không hỗ trợ thẻ audio.
  </audio>
</body>
</html>
```





Thẻ ngữ nghĩa (Semantic Tags)

Đây là một trong những cải tiến lớn nhất của HTML5. Các thẻ này không chỉ định dạng nội dung mà còn **truyền tải ý nghĩa** của nội dung đó. Điều này giúp cả con người (lập trình viên) và máy móc (trình duyệt, công cụ tìm kiếm) hiểu rõ hơn về cấu trúc của trang web.

Thay vì chỉ dùng thẻ `<div>` chung chung, các lập trình viên có thể dùng các thẻ sau để xây dựng cấu trúc trang web một cách rõ ràng hơn:

- **<header>**: Chứa phần đầu của trang hoặc một phần của trang, thường bao gồm logo, tiêu đề và thanh điều hướng.

- **<nav>**: Định nghĩa một phần của trang chứa các liên kết điều hướng.

- **<main>**: Chứa nội dung chính và độc lập của trang. Một trang chỉ nên có một thẻ `<main>`.

- **<article>**: Định nghĩa một phần nội dung độc lập và có ý nghĩa riêng, có thể đứng riêng lẻ như một bài blog, một bài báo.

- **<section>**: Định nghĩa một phần hoặc một chương của tài liệu, thường có tiêu đề riêng.

- **<aside>**: Định nghĩa nội dung liên quan nhưng không phải là nội dung chính, chẳng hạn như thanh bên (sidebar) hoặc các trích dẫn.

- **<footer>**: Chứa phần cuối của trang hoặc một phần của trang, thường chứa thông tin liên hệ, bản quyền.

Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Thẻ HTML5</title>
</head>
<body>
  <header>
    <h1>Tên Website</h1>
    <nav>
      <a href="/">Trang chủ</a> | <a href="/blog">Blog</a>
    </nav>
  </header>
  <main>
    <article>
      <h2>Tiêu đề bài viết</h2>
      <p>Nội dung chính của bài viết...</p>
    </article>
  </main>
  <footer>
    <p>© 2024 Bản quyền thuộc về tôi</p>
  </footer>
</body>
</html>
```

2.2. CSS (Cascading Style Sheets)

2.2.1. Giới thiệu CSS và vai trò trong thiết kế web

❖ CSS là gì?

CSS là chữ viết tắt của Cascading Style Sheets (tạm dịch là "các tệp định kiểu theo tầng"). Đây là một ngôn ngữ được sử dụng để định dạng và trình bày các tài liệu được viết bằng HTML. CSS giúp kiểm soát màu sắc, phông chữ, bố cục, khoảng cách và nhiều khía cạnh hình thức khác của một trang web.

Nếu bạn coi HTML là bộ xương của một trang web (xác định cấu trúc), thì CSS chính là lớp da, quần áo và trang sức (định dạng vẻ ngoài).

❖ Vai trò của CSS trong thiết kế web

- Tách biệt nội dung và định dạng: Đây là vai trò quan trọng nhất của CSS. Trước đây, việc định dạng được thực hiện trực tiếp trong HTML, làm cho mã nguồn trở nên rối rắm và khó quản lý. Với CSS, nội dung (HTML) và cách hiển thị (CSS) được tách biệt hoàn toàn. Điều này giúp:

+ Dễ bảo trì: Bạn có thể thay đổi giao diện của toàn bộ trang web chỉ bằng cách chỉnh sửa một vài dòng CSS mà không cần động đến nội dung HTML.

+ Tái sử dụng: Một tệp CSS có thể được sử dụng cho nhiều trang HTML khác nhau, giúp tiết kiệm thời gian và công sức.

- Tạo giao diện hấp dẫn và chuyên nghiệp: CSS biến một trang web thô sơ, chỉ có văn bản, thành một trang web có thiết kế đẹp mắt, hiện đại và thu hút người dùng.

- Cải thiện trải nghiệm người dùng (UX): Bố cục được sắp xếp hợp lý, màu sắc hài hòa và các hiệu ứng chuyển động mượt mà giúp người dùng dễ dàng tìm kiếm thông tin và tương tác với trang web hơn.

- Hỗ trợ đa thiết bị (Responsive Design): CSS có thể tự động điều chỉnh giao diện để trang web hiển thị tốt trên mọi kích thước màn hình, từ máy tính để bàn, máy tính bảng cho đến điện thoại di động.

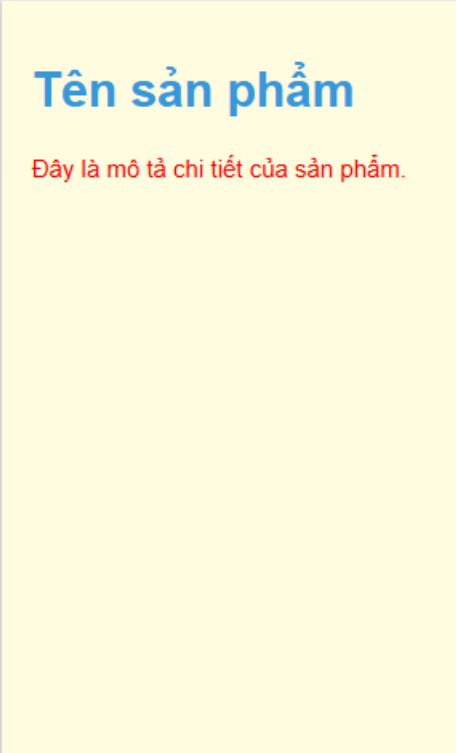
❖ Ví dụ về vai trò của CSS

Hãy tưởng tượng bạn có một trang HTML đơn giản. Nếu chỉ dùng HTML, bạn sẽ có một trang web cơ bản với văn bản đen trên nền trắng.

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>CSS</title>
</head>
<body>
  <h1>Tên sản phẩm</h1>
  <p>Đây là mô tả chi tiết của sản phẩm.</p>
  </body>
</html>
```

Khí HTML+CSS

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>CSS</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #FFFCDF;
        }
        h1 {
          color: #3498db;
          font-family: Arial, sans-serif;
        }
        p {
          color: red;
          line-height: 1.6;
        }
    </style>
</head>
<body>
  <h1>Tên sản phẩm</h1>
  <p>Đây là mô tả chi tiết của sản phẩm.</p>
  </body>
</html>
```



2.2.2. Các cách nhúng CSS vào HTML (inline, internal, external)

❖ CSS nội tuyến (Inline CSS)

- Cách thức: Viết mã CSS trực tiếp vào thẻ HTML bằng thuộc tính `style`.

- Ưu điểm: Nhanh chóng, đơn giản cho việc định dạng một phần tử duy nhất.

- Nhược điểm:

+ Khó quản lý: Nếu bạn cần định dạng nhiều phần tử giống nhau, bạn sẽ phải viết lại mã CSS nhiều lần.

+ Tách biệt kém: Vi phạm nguyên tắc tách biệt giữa nội dung (HTML) và định dạng (CSS), làm cho mã nguồn trở nên rối rắm.

- Khi nào sử dụng? Chỉ nên dùng cho những trường hợp đặc biệt, khi cần áp dụng một kiểu định dạng duy nhất cho một phần tử cụ thể.

Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>CSS</title>
</head>
<body>
  <p style="color: blue; font-size: 25px;">
      Đoạn văn định dạng bằng inline CSS.
  </p>
  </body>
</html>
```

Đoạn văn định dạng bằng inline CSS.

❖ CSS nội bộ (Internal CSS)

- Cách thức: Viết mã CSS trong một thẻ `<style>` được đặt bên trong phần `<head>` của tài liệu HTML.

- Ưu điểm:

+ Tập trung: Toàn bộ mã CSS của một trang được gom lại một chỗ, dễ dàng xem và chỉnh sửa.

+ Hiệu quả: Phù hợp khi bạn cần định dạng nhiều phần tử trên cùng một trang.

- Nhược điểm:

Không tái sử dụng: Chỉ áp dụng được cho trang HTML đó. Nếu bạn có nhiều trang, bạn sẽ phải sao chép mã CSS cho từng trang.

- Khi nào sử dụng? Khi bạn có một trang HTML duy nhất hoặc một trang có các kiểu định dạng đặc biệt, không cần áp dụng cho các trang khác.

- Ví dụ:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>CSS</title>
    <style>
        h1 {
            color: green;
        }
        p {
            font-family: Arial, sans-serif;
            font-size:17px;
        }
    </style>
</head>
<body>
    <h1>Đây là một tiêu đề được định dạng bằng
internal CSS.</h1>
    <p>Nội dung này cũng được định dạng bởi internal
CSS.</p>
  </body>
</html>
```

Đây là một tiêu đề được định dạng bằng internal CSS.

Nội dung này cũng được định dạng bởi internal CSS.

❖ CSS bên ngoài (External CSS)

- Cách thức: Tạo một tệp CSS riêng biệt (ví dụ: `style.css`) và liên kết nó với tài liệu HTML bằng thẻ `<link>` trong phần `<head>`. Đây là phương pháp phổ biến và được khuyến khích nhất.

- Ưu điểm:

+ Tái sử dụng: Một tệp CSS có thể được sử dụng cho nhiều trang web khác nhau.

+ Quản lý dễ dàng: Dễ dàng bảo trì và cập nhật. Khi bạn thay đổi một dòng mã trong tệp CSS, sự thay đổi đó sẽ được áp dụng cho tất cả các trang HTML có liên kết đến tệp đó.

+ Tách biệt hoàn toàn: Tách biệt hoàn toàn nội dung và định dạng, giúp mã nguồn HTML gọn gàng và dễ đọc hơn.

- Nhược điểm: Cần tải thêm một tệp riêng, nhưng lợi ích về quản lý và tái sử dụng lớn hơn nhiều.

- Khi nào sử dụng? Gần như mọi lúc, đặc biệt là trong các dự án web chuyên nghiệp.

- Ví dụ:

"mystyle.css"

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="mystyle.css">
</head>
<body>
  <h1>Đây là tiêu đề</h1>
  <p>Đây là đoạn văn bản</p>
</body>
</html>
```

2.2.3. Cú pháp CSS, bộ chọn (selectors), thuộc tính và giá trị

- Cú pháp CSS

Cú pháp CSS rất đơn giản và dễ hiểu. Nó bao gồm hai phần chính: **bộ chọn** và **khối khai báo**.

```
selector {
 property: value;
}
```

+ Bộ chọn (selector): Dùng để xác định phần tử HTML mà bạn muốn áp dụng các kiểu định dạng.

+ Khối khai báo (declaration block): Được bao quanh bởi dấu ngoặc nhọn {}. Khối này chứa một hoặc nhiều khai báo.

+ Khai báo (declaration): Bao gồm một cặp thuộc tính và giá trị, được phân tách bằng dấu hai chấm (:). Mỗi khai báo phải kết thúc bằng dấu chấm phẩy (;).

- Ví dụ:

```
h1 {
  text-align: center;
  color: red;
}
```

❖ **Bộ chọn (Selectors)**

Bộ chọn là một phần tử quan trọng trong CSS, giúp bạn "chọn" đúng phần tử HTML để định dạng. Có nhiều loại bộ chọn khác nhau:

- **Bộ chọn phần tử (Element Selector)**: Chọn tất cả các phần tử có cùng tên thẻ. Đây là cách cơ bản nhất.

```
p {
  text-align: center;
  color: red;
}
```

- **Bộ chọn ID (ID Selector)**: Chọn một phần tử duy nhất có thuộc tính `id` cụ thể. Bộ chọn này bắt đầu bằng dấu `#`.

```
#para1 {
  text-align: center;
  color: red;
}
```

- **Bộ chọn lớp (Class Selector)**: Chọn tất cả các phần tử có thuộc tính `class` cụ thể. Bộ chọn này bắt đầu bằng dấu (.). Đây là loại bộ chọn được sử dụng phổ biến nhất.

```
.intro {
  background-color: yellow;
}
```

- **Bộ chọn nhóm (Grouping Selector)**: Chọn nhiều phần tử cùng lúc bằng cách phân tách chúng bằng dấu phẩy (,).

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

- **Bộ chọn con cháu (Descendant Selector)**: Chọn các phần tử nằm bên trong một phần tử khác.

```
div p {
  background-color: yellow;
}
```

Sẽ áp dụng nền mầu vàng đối với toàn bộ các thẻ p năm trong thẻ div

❖ **Ví dụ tổng hợp**

```
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Bộ chọn phần tử */
        body {
            font-family: Arial, sans-serif;
        }
        /* Bộ chọn lớp */
        .highlight {
            background-color: yellow;
            font-weight: bold;
        }
        /* Bộ chọn ID */
        #main-title {
            color: #c0392b;
            text-align: center;
        }
        /* Bộ chọn con cháu */
        #content p {
            line-height: 1.5;
        }
    </style>
</head>
<body>
    <h1 id="main-title">Đây là tiêu đề chính</h1>
    <div id="content">
        <p class="highlight">Đoạn văn này được tô sáng.</p>
        <p>Đoạn văn này có khoảng cách dòng lớn hơn.</p>
    </div>
</body>
</html>
```

Đây là tiêu đề chính

Đoạn văn này được tô sáng.

Đoạn văn này có khoảng cách dòng lớn hơn.

2.2.4. Các nhóm thuộc tính CSS cơ bản

❖ **Nhóm thuộc tính định dạng văn bản**

Nhóm này tập trung vào việc tạo kiểu cho chữ viết, giúp nội dung dễ đọc và hấp dẫn hơn.

- **`font-family`**: Chọn phông chữ cho văn bản (ví dụ: `font-family: Arial, sans-serif;`).

- **`font-size`**: Điều chỉnh kích thước chữ (ví dụ: `font-size: 16px;`).

- **`font-weight`**: Đặt độ đậm của chữ (ví dụ: `font-weight: bold;`).

- **`color`**: Đặt màu chữ (ví dụ: `color: #333;`).

- **`text-align`**: Căn chỉnh văn bản theo chiều ngang (trái, phải, giữa, hoặc đều hai bên) (ví dụ: `text-align: center;`).

Ví dụ:

```
p {
  font-family: "Helvetica Neue", Arial, sans-serif;
  font-size: 1.1em;
  color: #444;
  text-align: justify;
}
```

❖ **Nhóm thuộc tính định dạng khung và nền**

Nhóm này kiểm soát các khía cạnh về nền và viền của một phần tử, giúp tạo ra các khối nội dung có tổ chức.

- **`border`**: Tạo viền xung quanh một phần tử. Bạn có thể định nghĩa độ dày, kiểu và màu sắc (ví dụ: `border: 1px solid #ccc;`).

- **`background-color`**: Đặt màu nền cho một phần tử (ví dụ: `background-color: #f5f5f5;`).

- **`background-image`**: Thêm một hình ảnh làm nền (ví dụ: `background-image: url('bg.jpg');`).

- **padding**: Tạo khoảng đệm (khoảng trống bên trong viền), đẩy nội dung ra xa viền (ví dụ: `padding: 15px;`).

- **margin**: Tạo khoảng lề (khoảng trống bên ngoài viền), đẩy phần tử ra xa các phần tử khác (ví dụ: `margin: 20px;`).

Ví dụ:

```
.card {
  border: 2px solid #3498db;
  background-color: #ecf0f1;
  padding: 20px;
  margin: 10px;
}
```

❖ **Nhóm thuộc tính định dạng bố cục**

Đây là nhóm thuộc tính quan trọng nhất để xây dựng cấu trúc và sắp xếp các phần tử trên trang web.

- **display**: Thuộc tính này xác định cách một phần tử được hiển thị.

+ **block**: Mỗi phần tử bắt đầu trên một dòng mới và chiếm toàn bộ chiều rộng. Ví dụ: `div`, `p`, `h1`.

+ **inline**: Phần tử chỉ chiếm chiều rộng cần thiết và không bắt đầu trên dòng mới. Ví dụ: `span`, `a`, `img`.

+ **flex** và **grid**: Các giá trị hiện đại giúp tạo ra bố cục phức tạp một cách linh hoạt, đặc biệt hiệu quả trong việc sắp xếp các phần tử.

- **position**: Kiểm soát vị trí của một phần tử.

+ **static**: Vị trí mặc định.

+ **relative**: Vị trí tương đối so với vị trí ban đầu.

+ **absolute**: Vị trí tuyệt đối so với phần tử bao bọc gần nhất có `position` khác `static`.

+ **fixed**: Vị trí cố định so với cửa sổ trình duyệt (ví dụ: thanh menu luôn hiển thị).

- **float**: Dùng để đẩy một phần tử sang bên trái hoặc bên phải, cho phép các phần tử khác bao quanh nó. Thường được dùng để tạo bố cục có hình ảnh bên cạnh văn bản.

Ví dụ:

```
/* Sử dụng Flexbox để căn giữa nội dung */
.container {
    display: flex;
    justify-content: center;
    align-items: center;
}
/* Sử dụng position để đặt một logo ở góc trên cùng bên trái */
.logo {
    position: fixed;
    top: 10px;
    left: 10px;
}
```

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Demo Flexbox và Position</title>
    <style>
        /* Sử dụng Flexbox để căn giữa nội dung */
        .container {
            display: flex;
            justify-content: center;
            align-items: center;
        }
        /* Sử dụng position để đặt một logo ở góc trên cùng bên trái */
        .logo {
            position: fixed;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div class="logo">
        <p>Logo</p>
    </div>
    <div class="container">
        <p>Đây là box được căn giữa</p>
    </div>
</body>
</html>
```

Logo

Đây là box được căn giữa

2.2.5. Pseudo-class, Pseudo-element và kết hợp selectors

❖ Pseudo-class

Pseudo-class (lớp giả) là một từ khóa được thêm vào bộ chọn (selector) để xác định một trạng thái đặc biệt của một phần tử. Nó cho phép bạn áp dụng kiểu dáng cho một phần tử khi nó ở một trạng thái cụ thể mà không cần phải thay đổi mã HTML.

- Cú pháp: `selector:pseudo-class { property: value; }`

- Dấu hiệu nhận biết: Luôn bắt đầu bằng dấu hai chấm (`:`)

Các Pseudo-class phổ biến:

`:hover`: Áp dụng kiểu khi con trỏ chuột di chuyển qua phần tử.

`:active`: Áp dụng kiểu khi phần tử được nhấn giữ.

`:focus`: Áp dụng kiểu khi phần tử được chọn (ví dụ: khi click vào một ô input).

`:visited`: Áp dụng kiểu cho các liên kết đã được người dùng truy cập.

`:first-child`: Áp dụng kiểu cho phần tử con đầu tiên trong một nhóm.

Ví dụ:

```css
a:hover {
  color: red; /* Chữ chuyển sang màu đỏ khi di chuột qua */
}
input:focus {
  border: 2px solid blue; /* Viền xanh khi ô input được chọn */
}
```

❖ 2. Pseudo-element

Pseudo-element (phần tử giả) là từ khóa được thêm vào bộ chọn để xác định một phần của một phần tử. Nó cho phép bạn tạo kiểu cho một phần cụ thể của nội dung mà không cần phải thêm thẻ HTML mới.

- Cú pháp: `selector::pseudo-element { property: value; }`

- Dấu hiệu nhận biết: Luôn bắt đầu bằng hai dấu hai chấm `::` (mặc dù một số trình duyệt cũ vẫn chấp nhận một dấu hai chấm).

Các Pseudo-element phổ biến:

`::before`: Chèn nội dung ảo vào **trước** nội dung của phần tử.

`::after`: Chèn nội dung ảo vào **sau** nội dung của phần tử.

`::first-line`: Áp dụng kiểu cho dòng đầu tiên của một đoạn văn bản.

`::first-letter`: Áp dụng kiểu cho chữ cái đầu tiên của một đoạn văn bản.

Ví dụ:

```
<!DOCTYPE html>
<html>
<head>
<style>
  p::first-letter {
    font-size: 200%;/* Phóng to chữ cái đầu tiên */
    font-weight: bold;
    color: #8A2BE2;
  }
  h2::after {
    content: " (mới)"; /* Thêm chữ "(mới)" sau tiêu đề */
    color: green;
  }
</style>
</head>
<body>
  <p>Phạm Phương Linh.</p>
  <h2>Mình mới mua một bộ váy</h2>
</body>
</html>
```

Phạm Phương Linh.

Mình mới mua một bộ váy (mới)

❖ Kết hợp selectors

Bạn có thể kết hợp các bộ chọn để tạo ra các quy tắc CSS cụ thể và mạnh mẽ hơn.

- Kết hợp Pseudo-class với Element/Class:

`button.primary:hover { ... }`: Áp dụng kiểu khi di chuột qua nút bấm có class là `primary`.

- Kết hợp Pseudo-class và Pseudo-element:

`a:hover::before { ... }`: Chèn nội dung ảo vào trước liên kết khi di chuột qua nó.

Ví dụ tổng hợp:

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <title>Pseudo-class, Pseudo-element</title>
    <style>
        .product-link:hover {
            color: #e74c3c;
            text-decoration: underline;
        }
        .product-link:hover::after {
            content: " 🚀"; /* Thêm icon sau liên kết khi di chuột */
        }
        .intro p::first-line {
            font-style: italic; /* Dòng đầu tiên của đoạn văn trong intro sẽ in nghiêng */
        }
    </style>
</head>
<body>
    <div class="intro">
        <p>Đây là một đoạn văn bản giới thiệu. Dòng đầu tiên sẽ được định dạng khác.</p>
    </div>
    <a href="#" class="product-link">Xem chi tiết sản phẩm</a>
</body>
</html>
```

Đây là một đoạn văn bản giới thiệu. Dòng đầu tiên sẽ được định dạng khác.

Xem chi tiết sản phẩm 🚀

2.2.6. Responsive Design

❖ Responsive Design là gì?

Responsive Design (Thiết kế đáp ứng) là một phương pháp thiết kế và lập trình web, cho phép giao diện của trang web tự động điều chỉnh để hiển thị tối ưu trên mọi kích thước màn hình và thiết bị, từ máy tính để bàn có màn hình lớn, máy tính bảng cho đến điện thoại di động.

Trước đây, các lập trình viên thường phải tạo ra các phiên bản website riêng biệt cho từng loại thiết bị. Với Responsive Design, chỉ cần một trang web duy nhất có thể "đáp ứng" với môi trường của người dùng, mang lại trải nghiệm nhất quán và hiệu quả.

❖ Media Queries

Media Queries là kỹ thuật cốt lõi của Responsive Design. Nó là một tính năng trong CSS3, cho phép bạn áp dụng các kiểu định dạng khác nhau dựa trên các đặc điểm của thiết bị mà người dùng đang sử dụng, chẳng hạn như chiều rộng màn hình, chiều cao, hoặc độ phân giải.

- Cú pháp: Media Queries được viết trong một khối `@media` theo sau là một điều kiện.

```
@media (điều kiện) {
  /* Các quy tắc CSS sẽ được áp dụng khi điều kiện đúng */
}
```

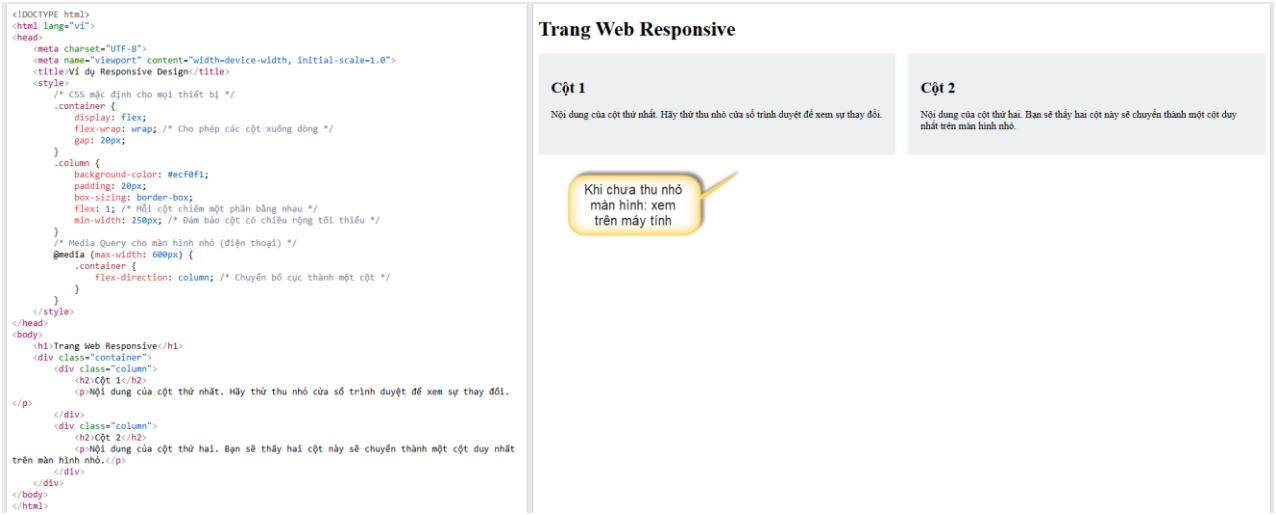
- Các điều kiện phổ biến:

`max-width`: Áp dụng kiểu CSS khi chiều rộng màn hình **tối đa** là một giá trị cụ thể. Thường được sử dụng để định nghĩa kiểu cho thiết bị di động.

`min-width`: Áp dụng kiểu CSS khi chiều rộng màn hình **tối thiểu** là một giá trị cụ thể. Thường được dùng để định nghĩa kiểu cho máy tính bảng hoặc máy tính bàn.

- Ví dụ về cách sử dụng Media Queries: Hãy tưởng tượng bạn có một trang web với hai cột nội dung trên máy tính để bàn. Khi xem trên điện thoại, bạn muốn các cột này chuyển thành một cột duy nhất để dễ đọc hơn.

Chế độ xem ở màn hình rộng hơn 600px



Chế độ xem ở màn hình nhỏ hơn hoặc bằng 600px

Trong ví dụ trên, quy tắc CSS trong `@media` chỉ được áp dụng khi chiều rộng màn hình nhỏ hơn hoặc bằng 600px. Điều này giúp trang web tự động điều chỉnh bố cục một cách linh hoạt, tạo trải nghiệm tốt hơn cho người dùng trên mọi thiết bị.

2.3. JavaScript Cơ Bản

2.3.1. Giới thiệu JavaScript và vai trò phía Client

❖ **Giới thiệu JavaScript**

JavaScript (JS) là một ngôn ngữ lập trình kịch bản (scripting language) mạnh mẽ, được sử dụng chủ yếu để tạo ra các trang web tương tác. Khác với HTML (ngôn ngữ đánh dấu) và CSS (ngôn ngữ định kiểu), JavaScript cho phép bạn thêm **hành vi động** vào trang web, biến một trang web tĩnh thành một ứng dụng sống động.

- **Tên gọi:** Ban đầu, JavaScript được gọi là LiveScript, sau đó được đổi tên thành JavaScript để tận dụng sự phổ biến của Java vào thời điểm đó. Tuy nhiên, JavaScript và Java là hai ngôn ngữ hoàn toàn khác nhau.

- **Vị trí:** JavaScript hoạt động chủ yếu ở phía máy khách (Client-side), nghĩa là mã được chạy trực tiếp trên trình duyệt của người dùng, không phải trên máy chủ.

❖ **Vai trò của JavaScript ở phía Client**

JavaScript đóng vai trò "bộ não" của trang web, cho phép trang web thực hiện các hành động thông minh mà HTML và CSS không thể làm được.

- **Tạo tương tác người dùng (User Interaction):** JavaScript giúp trang web phản ứng với các hành động của người dùng như click chuột, di chuyển con trỏ, hoặc nhập liệu.

Ví dụ: Khi bạn click vào một nút, một cửa sổ popup hiện ra; hoặc khi bạn điền một form, JavaScript sẽ kiểm tra xem các thông tin đó có hợp lệ không.

- **Thao tác với DOM (Document Object Model):** JavaScript có khả năng truy cập và thay đổi cấu trúc, nội dung và kiểu dáng của trang web một cách linh hoạt.

Ví dụ: JavaScript có thể thêm, xóa, hoặc chỉnh sửa các phần tử HTML (như một đoạn văn bản hoặc một hình ảnh) mà không cần tải lại trang.

- **Xử lý dữ liệu và logic:** JavaScript thực hiện các phép tính, xử lý dữ liệu từ form, tạo các hiệu ứng động, và quản lý các sự kiện.

Ví dụ: Khi bạn nhập một sản phẩm vào ô tìm kiếm, JavaScript có thể lọc và hiển thị kết quả ngay lập tức mà không cần gửi yêu cầu đến máy chủ.

- **Giao tiếp bất đồng bộ (Asynchronous Communication):** JavaScript có thể gửi và nhận dữ liệu từ máy chủ mà không làm gián đoạn trải nghiệm của người dùng.

Ví dụ: Các ứng dụng như Facebook và Gmail sử dụng JavaScript để tải nội dung mới (như tin nhắn, thông báo) mà không cần phải tải lại toàn bộ trang. Kỹ thuật này được gọi là AJAX.

Tóm lại, nếu **HTML** là **cấu trúc** của một trang web và **CSS** là **vẻ ngoài**, thì **JavaScript** chính là **hành vi** của nó. JavaScript mang lại sự sống và sự tương tác cho trang web, biến nó từ một tài liệu tĩnh thành một ứng dụng web năng động.

2.3.2. Cách nhúng JavaScript vào HTML (internal, external)

❖ JavaScript nội bộ (Internal JavaScript)

Cách thức: Viết mã JavaScript trực tiếp vào trong thẻ `<script>` và đặt thẻ này trong phần `<head>` hoặc `<body>` của tài liệu HTML.

- Ưu điểm:

+ Nhanh chóng và tiện lợi: Thích hợp cho các tệp HTML đơn giản hoặc khi bạn muốn thực hiện các chức năng nhỏ, cụ thể chỉ trên một trang.

+ Không cần tệp tin bên ngoài: Mọi thứ được chứa trong một tệp duy nhất, không cần phải quản lý nhiều tệp tin.

- Nhược điểm:

+ Khó tái sử dụng: Bạn không thể sử dụng lại mã này cho các trang HTML khác mà không sao chép lại.

+ Làm cho mã HTML cồng kềnh: Nếu mã JavaScript quá dài, nó sẽ khiến tệp HTML của bạn trở nên rối rắm và khó đọc.

- Khi nào sử dụng? Khi bạn có một đoạn mã JavaScript nhỏ, cụ thể chỉ dành cho một trang.

- Ví dụ: Javascript ở <body>

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Internal JavaScript</title>
</head>
<body>
    <h1>Chào bạn!</h1>
    <button onclick="showAlert()">Nhấn vào đây</button>
    <script>
        function showAlert() {
            alert('Bạn vừa nhấn vào nút!');
        }
    </script>
</body>
</html>
```

- Ví dụ: Javascript ở <header>

```
<!DOCTYPE html>
<html>
<head>
<script>
  function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
  }
  </script>
</head>
<body>
  <h2>Demo JavaScript in Head</h2>
  <p id="demo">A Paragraph.</p>
  <button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

❖ **JavaScript bên ngoài (External JavaScript)**

- **Cách thức:** Tạo một tệp JavaScript riêng biệt (ví dụ: `script.js`) và liên kết nó với tệp HTML bằng cách sử dụng thẻ `<script>` với thuộc tính `src`. Đây là phương pháp phổ biến và được khuyến khích nhất.

- Ưu điểm:

+ Tái sử dụng: Một tệp JavaScript có thể được liên kết và sử dụng trên nhiều trang HTML khác nhau.

+ Dễ quản lý: Tách biệt mã JavaScript ra khỏi HTML và CSS, giúp mã nguồn trở nên sạch sẽ và có tổ chức hơn.

+ Tăng tốc độ tải trang: Trình duyệt có thể lưu trữ tệp JavaScript vào bộ nhớ cache, giúp các lần truy cập sau đó nhanh hơn.

- Nhược điểm: Cần tải thêm một tệp riêng, nhưng lợi ích về quản lý và hiệu năng lớn hơn nhiều.

- Khi nào sử dụng? Gần như mọi lúc, đặc biệt là trong các dự án lớn, chuyên nghiệp.

- Ví dụ: `script.js` bên ngoài file index.html

```
function showAlert() {
    alert('Bạn vừa nhấn vào nút từ tệp JavaScript bên ngoài!');
}
```

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>External JavaScript</title>
</head>
<body>

    <h1>Chào bạn!</h1>
    <button onclick="showAlert()">Nhấn vào đây</button>
    <script src="script.js"></script>

</body>
</html>
```

2.3.3. Cấu trúc cú pháp và kiểu dữ liệu

❖ **Cấu trúc cú pháp**

Cấu trúc cú pháp của JavaScript khá đơn giản, dựa trên ngôn ngữ C và Java. Mỗi câu lệnh trong JavaScript đều được thực thi theo thứ tự từ trên xuống dưới.

- Sử dụng dấu chấm phẩy (`;`): Mỗi câu lệnh thường kết thúc bằng dấu chấm phẩy. Mặc dù JavaScript có thể tự động thêm dấu chấm phẩy trong một số trường hợp, việc sử dụng nó là một thói quen tốt để tránh các lỗi không mong muốn.

```
let x = 10;
let y = 20;
```

- Khai báo biến: Sử dụng các từ khóa như `var`, `let`, và `const`.

`var`: Cách khai báo cũ, có phạm vi hoạt động (scope) rộng.

`let`: Dùng để khai báo biến có thể thay đổi giá trị, có phạm vi khối (block scope).

`const`: Dùng để khai báo hằng số, giá trị không thể thay đổi.

- Ghi chú (Comments): Dùng để giải thích mã nguồn. Trình duyệt sẽ bỏ qua các dòng này.

+ Ghi chú một dòng: Sử dụng `//`.

+ Ghi chú nhiều dòng: Sử dụng `/* ... */`.

- Phân biệt chữ hoa/chữ thường (Case-sensitive): JavaScript phân biệt chữ hoa và chữ thường. Ví dụ, `myVariable` và `myvariable` là hai biến khác nhau.

❖ Các kiểu dữ liệu cơ bản

JavaScript có nhiều kiểu dữ liệu khác nhau để lưu trữ các loại thông tin. Dưới đây là các kiểu dữ liệu nguyên thủy (primitive) phổ biến:

- **String (Chuỗi)**: Dùng để lưu trữ văn bản. Chuỗi được đặt trong dấu ngoặc kép (`"..."`) hoặc ngoặc đơn (`'...'`).

Ví dụ: `let name = "John";`

- **Number (Số)**: Dùng để lưu trữ các giá trị số, bao gồm cả số nguyên và số thập phân.

Ví dụ: `let age = 30;` hoặc `let price = 19.99;`

- **Boolean (Kiểu luận lý)**: Dùng để lưu trữ hai giá trị: `true` (đúng) hoặc `false` (sai). Thường được sử dụng trong các câu lệnh điều kiện.

Ví dụ: `let isStudent = true;`

- **Null**: Dùng để biểu thị một giá trị **rỗng** hoặc **không có giá trị**.

Ví dụ: `let car = null;`

- **Undefined**: Biểu thị một biến đã được khai báo nhưng **chưa được gán giá trị**.

Ví dụ: `let address;` // biến `address` sẽ có giá trị là `undefined`

- **Object (Đối tượng)**: Một kiểu dữ liệu phức tạp hơn, dùng để lưu trữ một tập hợp các thuộc tính và giá trị. Đây là một khái niệm rất quan trọng trong JavaScript.

```
<script>
   let person = {
      firstName: "Jane",
      lastName: "Doe",
      age: 25
  };
</script>
```

- **Array (Mảng)**: Một kiểu dữ liệu đặc biệt của đối tượng, dùng để lưu trữ một danh sách các giá trị được sắp xếp

```
<script>
    let colors = ["red", "green", "blue"];
</script>
```

2.3.4. Biến, hằng, toán tử, cấu trúc điều khiển

❖ 1. Biến (Variables) và Hằng (Constants)

Biến là nơi lưu trữ dữ liệu, có thể thay đổi giá trị trong suốt quá trình chạy chương trình. **Hằng** cũng là nơi lưu trữ dữ liệu, nhưng giá trị của nó **không thể thay đổi** sau khi được gán.

- Từ khóa khai báo:

`var`: Cách khai báo cũ, có thể thay đổi giá trị. Nó có phạm vi hoạt động (scope) là hàm (function scope).

`let`: Cách khai báo hiện đại, có thể thay đổi giá trị. Nó có phạm vi hoạt động là khối (block scope), an toàn hơn `var`.

`const`: Dùng để khai báo hằng số. Nó cũng có phạm vi khối.

- Ví dụ:

```
<script>
    let age = 25; // Biến: có thể thay đổi
    age = 26;
    const PI = 3.14; // Hằng: không thể thay đổi
    // PI = 3.15; // Lỗi: TypeError
</script>
```

❖ Toán tử (Operators)

Toán tử được sử dụng để thực hiện các phép toán hoặc so sánh trên dữ liệu.

- **Toán tử số học:** Dùng để tính toán (+, -, *, /, %, ++, --).

- **Toán tử gán:** Dùng để gán giá trị (=, +=, -=, *=, /=).

- **Toán tử so sánh:** So sánh hai giá trị, trả về `true` hoặc `false` (==, ===, !=, !==, <, >, <=, >=).

`==` (bằng): Chỉ so sánh giá trị, không so sánh kiểu dữ liệu. Ví dụ: `'5' == 5` là `true`.

`===` (bằng tuyệt đối): So sánh cả giá trị và kiểu dữ liệu. Ví dụ: `'5' === 5` là `false`.

- **Toán tử logic:** Kết hợp các biểu thức boolean (&&, ||, !).

`&&` (và): Cả hai điều kiện đều phải đúng.

`||` (hoặc): Một trong hai điều kiện đúng.

`!` (phủ định): Đảo ngược giá trị boolean.

❖ Cấu trúc điều khiển (Control Structures)

Cấu trúc điều khiển giúp bạn quyết định hướng đi của chương trình dựa trên các điều kiện cụ thể.

- Câu lệnh điều kiện `if/else`

Cho phép thực thi một đoạn mã nếu một điều kiện là đúng, và một đoạn mã khác nếu điều kiện là sai.

Cú pháp:

```
<script>
    if (điều_kiện) {
        // Mã sẽ chạy nếu điều kiện đúng
    } else {
        // Mã sẽ chạy nếu điều kiện sai
    }
</script>
```

Ví dụ:

```
<script>
  let score = 85;
  if (score >= 50) {
      console.log("Bạn đã đỗ.");
  } else {
      console.log("Bạn đã trượt.");
  }
</script>
```

- Cấu trúc lặp `for`

Thực thi một đoạn mã lặp đi lặp lại một số lần xác định.

Cú pháp:

```
<script>
    for (khởi_tạo; điều_kiện; bước_nhảy) {
        // Mã sẽ chạy trong mỗi vòng lặp
    }
</script>
```

Ví dụ:

```
<script>
  for (let i = 0; i < 5; i++) {
      console.log("Số: " + i);
  }
</script>
```

- Cấu trúc lặp `while`

Thực thi một đoạn mã lặp đi lặp lại miễn là điều kiện vẫn còn đúng.

```
<script>
    while (điều_kiện) {
        // Mã sẽ chạy miễn là điều kiện đúng
    }
</script>
```

Ví dụ:

```
<script>
    let i = 0;
    while (i < 3) {
        console.log("Lặp while lần " + i);
        i++;
    }
</script>
```

2.3.5. Hàm và phạm vi biến

❖ Hàm (Functions)

Hàm là một khối mã được thiết kế để thực hiện một nhiệm vụ cụ thể. Việc sử dụng hàm giúp mã nguồn của bạn có tổ chức, dễ tái sử dụng và dễ bảo trì hơn.

- **Cú pháp:** Bạn khai báo một hàm bằng từ khóa `function`, theo sau là tên hàm, danh sách các tham số (nếu có), và khối mã nằm trong dấu ngoặc nhọn `{}`.

```
<script>
    function ten_ham(tham_so_1, tham_so_2) {
      // Khối mã sẽ được thực thi
      // Có thể có câu lệnh return
    }
</script>
```

- **Cách sử dụng:** Để chạy hàm, bạn chỉ cần gọi tên hàm và truyền các đối số (arguments) tương ứng với các tham số.

- Lợi ích:

+ Tái sử dụng: Viết mã một lần và có thể gọi lại nhiều lần ở nhiều nơi khác nhau.

+ Tổ chức: Chia nhỏ chương trình thành các phần nhỏ hơn, dễ hiểu và quản lý.

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>
<p id="demo"></p>
<p id="demo1"></p>

<script>
  let x = myFunction(4, 3);
  let chao = chaoMung("Phạm Phương Linh");
  document.getElementById("demo").innerHTML ="Kết quả nhân 2 số: "+x;
  document.getElementById("demo1").innerHTML =chao;

  function myFunction(a, b) {
    return a * b;
  }

  function chaoMung(ten) {
   return "Xin chào, " + ten + "!";
  }
</script>
</body>
</html>
```

JavaScript Functions

Kết quả nhân 2 số: 12

Xin chào, Phạm Phương Linh!

- Lợi ích:

+ Tái sử dụng: Viết mã một lần và có thể gọi lại nhiều lần ở nhiều nơi khác nhau.

+ Tổ chức: Chia nhỏ chương trình thành các phần nhỏ hơn, dễ hiểu và quản lý.

❖ **Phạm vi biến (Variable Scope)**

Phạm vi biến là nơi mà biến đó có thể được truy cập trong chương trình. Hiểu rõ phạm vi giúp bạn tránh được các lỗi không đáng có và quản lý bộ nhớ hiệu quả. Trong JavaScript, có ba loại phạm vi chính:

- Phạm vi toàn cục (Global Scope):

+ Các biến được khai báo bên ngoài tất cả các hàm và khối mã.

+ Chúng có thể được truy cập từ bất kỳ đâu trong chương trình.

+ Ví dụ:

```
let globalVar = "Tôi là biến toàn cục";
function demoGlobal() {
  console.log(globalVar); // Có thể truy cập
}
```

- Phạm vi hàm (Function Scope) :

+ Các biến được khai báo bên trong một hàm bằng từ khóa `var`.

+ Chúng chỉ có thể được truy cập từ bên trong hàm đó.

+ Ví dụ:

```
function demoFunction() {
  var functionVar = "Tôi là biến trong hàm";
  console.log(functionVar); // Có thể truy cập
}
// console.log(functionVar); // Lỗi: ReferenceError
```

- Phạm vi khối (Block Scope):

+ Các biến được khai báo bên trong một khối mã (ví dụ: `if`, `for`, hoặc đơn giản là một cặp dấu `{}`) bằng từ khóa `let` hoặc `const`.

+ Chúng chỉ có thể được truy cập từ bên trong khối mã đó. Đây là phạm vi hiện đại và an toàn hơn.

+ Ví dụ:

```
if (true) {
  let blockVar = "Tôi là biến trong khối";
  console.log(blockVar); // Có thể truy cập
}
// console.log(blockVar); // Lỗi: ReferenceError
```

❖ **So sánh `var`, `let`, và `const`:**

- **`var`** có phạm vi hàm, dễ gây lỗi ngoài ý muốn khi biến được tái sử dụng trong các khối lồng nhau.

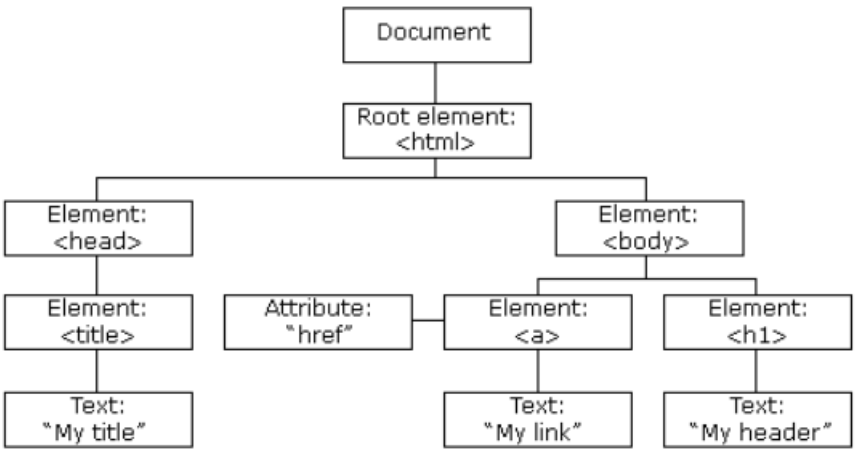
- **`let`** và **`const`** có phạm vi khối, giúp giới hạn phạm vi của biến, làm cho mã nguồn an toàn và dễ quản lý hơn. Đây là lý do tại sao `let` và `const` được khuyến khích sử dụng trong các dự án hiện đại.

2.3.6. DOM (Document Object Model) và thao tác với phần tử HTML

❖ **DOM (Document Object Model) là gì?**

DOM (Document Object Model - Mô hình Đối tượng Tài liệu) là một giao diện lập trình ứng dụng (API) cho các tài liệu HTML. Khi trình duyệt tải một trang web, nó sẽ xây dựng một cấu trúc cây (tree structure) đại diện cho toàn bộ trang đó. Mỗi thẻ HTML, mỗi đoạn văn bản, hay thậm chí mỗi thuộc tính, đều trở thành một nút (**node**) trong cấu trúc cây này.

Bạn có thể hình dung DOM giống như **bản đồ chi tiết** của một ngôi nhà. Ngôi nhà (trang web) có nhiều phòng (các phần tử như `<div>`, `<h1>`), và bản đồ (DOM) cung cấp cho bạn cách để tìm đến từng phòng, xem nội dung bên trong, và thậm chí thay đổi chúng.



❖ **Vai trò của DOM trong JavaScript**

JavaScript sử dụng DOM để **truy cập, thao tác và thay đổi** cấu trúc, nội dung và kiểu dáng của các phần tử HTML. Nhờ có DOM, JavaScript có thể:

- Thay đổi nội dung của một phần tử.
- Thay đổi kiểu dáng (CSS) của một phần tử.
- Thêm, xóa hoặc chỉnh sửa các phần tử HTML.
- Phản ứng lại các sự kiện của người dùng (như click chuột, nhấn phím).

❖ **Thao tác với phần tử HTML**

Để thao tác với một phần tử, bạn cần phải tìm và chọn phần tử đó trước. JavaScript cung cấp nhiều phương thức để thực hiện việc này.

- Chọn phần tử (Selecting Elements)

`document.getElementById('id-name')`: Tìm và trả về một phần tử duy nhất có thuộc tính `id` cụ thể. Đây là cách nhanh và trực tiếp nhất để chọn một phần tử.

`document.getElementsByClassName('class-name')`: Tìm và trả về một tập hợp các phần tử có cùng thuộc tính `class`.

`document.getElementsByTagName('tag-name')`: Tìm và trả về một tập hợp các phần tử có cùng tên thẻ.

`document.querySelector('selector')`: Trả về phần tử đầu tiên khớp với bộ chọn CSS.

`document.querySelectorAll('selector')`: Trả về tất cả các phần tử khớp với bộ chọn CSS.

- Thay đổi nội dung và thuộc tính

Sau khi đã chọn được phần tử, bạn có thể thay đổi nó.

`element.innerHTML`: Thay đổi hoặc lấy nội dung HTML bên trong một phần tử. Rất mạnh mẽ nhưng cần cẩn thận với bảo mật (ngăn chặn XSS).

`element.textContent`: Thay đổi hoặc lấy nội dung văn bản thuần túy của một phần tử.

`element.attribute = 'value'`: Thay đổi giá trị của một thuộc tính.

Ví dụ: `image.src = 'new_image.jpg';`

- Thay đổi kiểu dáng (CSS)

`element.style.property = 'value'`: Thay đổi kiểu dáng CSS của một phần tử bằng cách sử dụng thuộc tính `style`.

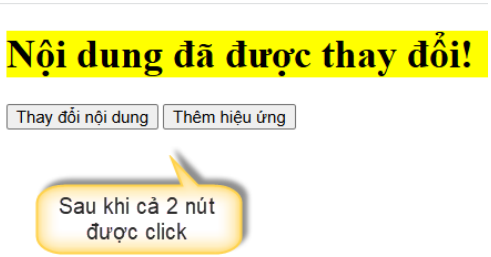
Ví dụ: `myElement.style.color = 'red';`

`element.classList.add('class-name')`: Thêm một lớp CSS vào phần tử.

`element.classList.remove('class-name')`: Xóa một lớp CSS khỏi phần tử.

❖ **Ví dụ tổng hợp:**

```html
<!DOCTYPE html>
<html>
<head>
    <style>
        .highlight {
            background-color: yellow;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1 id="main-title">Xin chào thế giới!</h1>
    <button onclick="changeText()">Thay đổi nội dung</button>
    <button onclick="addHighlight()">Thêm hiệu ứng</button>

    <script>
        function changeText() {
            //Trả về phần tử h1 vì có thuộc tính id=main-title
            const titleElement = document.getElementById('main-title');
            //Thay đổi phần tử h1
            titleElement.textContent = "Nội dung đã được thay đổi!";
        }

        function addHighlight() {
            //Trả về phần tử h1 vì có thuộc tính id=main-title
            const titleElement = document.getElementById('main-title');
            //Thêm một lớp highlight
            titleElement.classList.add('highlight');
        }
    </script>
</body>
</html>
```



2.3.7. Xử lý sự kiện (event handling)

❖ Xử lý sự kiện (Event Handling) là gì?

Sự kiện (Event) là những hành động hoặc những điều xảy ra trên trang web mà trình duyệt có thể phát hiện và phản ứng lại. Ví dụ về các sự kiện phổ biến bao gồm:

- Người dùng **click** chuột vào một nút.
- Người dùng **di chuyển** con trỏ chuột.
- Người dùng **nhấn phím** trên bàn phím.
- Trang web **tải xong**.
- Một ô input thay đổi giá trị.

Xử lý sự kiện (Event Handling) là quá trình sử dụng JavaScript để lắng nghe các sự kiện này và thực hiện một hành động cụ thể khi chúng xảy ra.

Bạn có thể hình dung việc này giống như một người phục vụ nhà hàng. Người phục vụ (chương trình JavaScript) luôn luôn lắng nghe khách hàng (các sự kiện). Khi khách hàng gọi món (sự kiện `click`), người phục vụ sẽ ghi lại yêu cầu và chuyển đến nhà bếp (thực hiện một hàm).

❖ Các cách xử lý sự kiện trong JavaScript

Có ba cách phổ biến để xử lý sự kiện trong JavaScript.

1. Xử lý sự kiện trực tiếp trong HTML (Inline Event Handling)

Đây là cách đơn giản nhất, nhưng không được khuyến khích trong các dự án chuyên nghiệp vì nó làm lẫn lộn HTML và JavaScript. Bạn sẽ thêm một thuộc tính sự kiện trực tiếp vào thẻ HTML.

- Cú pháp: `<tag attribute="JavaScript code">`
- Ví dụ:

```html
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>

<h2 onclick="this.innerHTML='Nội dung sau khi Click'">Click vào đây!</h2>
<button onclick="alert('Bạn đã CLick vào nút này!');">Click vào tôi</button>

</body>
</html>
```

2. Gán sự kiện bằng DOM Property

Đây là cách phổ biến hơn, bạn chọn phần tử bằng JavaScript và sau đó gán một hàm xử lý sự kiện cho thuộc tính sự kiện của nó.

- Cú pháp: `element.onevent = function_name;`

- Ví dụ:

```html
<!DOCTYPE html>
<html>
<body>
    <button id="myButton">Nhấp vào tôi</button>
    <script>
        const button = document.getElementById('myButton');
        function handleButtonClick() {
            alert('Bạn đã nhấp vào nút này!');
        }
        button.onclick = handleButtonClick;
    </script>
</body>
</html>
```

3. Sử dụng `addEventListener()`

Đây là phương pháp **hiện đại và được khuyến khích nhất** vì nó linh hoạt và mạnh mẽ hơn. Nó cho phép bạn thêm nhiều trình xử lý sự kiện cho cùng một phần tử, không giống như cách thứ hai chỉ cho phép một.

- Cú pháp: `element.addEventListener(event, function_name);`

- Ví dụ:

```html
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Xử lý sự kiện với thay đổi màu</title>
    <style>
        #myButton {
            padding: 10px 20px;
            font-size: 16px;
            background-color: #3498db;
            color: white;
            border: none;
            cursor: pointer;
            transition: background-color 0.3s ease; /* Tạo hiệu ứng chuyển đổi màu mượt mà */
        }
    </style>
</head>
<body>
    <button id="myButton">Nhấp vào tôi</button>
    <script>
        const button = document.getElementById('myButton');

        // Thêm trình xử lý sự kiện 'click'
        button.addEventListener('click', function() {
            alert('Bạn đã nhấp vào nút này!');
        });
        // Thêm trình xử lý sự kiện 'mouseover' để đổi màu khi di chuột vào
        button.addEventListener('mouseover', function() {
            button.style.backgroundColor = '#2ecc71'; // Đổi màu nền thành xanh lá
        });
        // Thêm trình xử lý sự kiện 'mouseout' để đổi màu về ban đầu khi di chuột ra
        button.addEventListener('mouseout', function() {
            button.style.backgroundColor = '#3498db'; // Đổi màu nền về xanh dương
        });
    </script>
</body>
</html>
```

Nhấp vào tôi

2.3.8. Kiểm tra và xử lý dữ liệu Form

❖ Kiểm tra và xử lý dữ liệu Form

Trong lập trình web, việc kiểm tra dữ liệu từ form là một bước quan trọng để đảm bảo tính hợp lệ của thông tin mà người dùng nhập vào. Quá trình này được gọi là **Validation**. JavaScript cho phép bạn thực hiện validation ngay tại phía máy khách (Client-side), giúp phản hồi ngay lập tức cho người dùng và giảm tải cho máy chủ.

Tại sao cần Validation?

- **Tăng trải nghiệm người dùng:** Phát hiện lỗi ngay khi người dùng nhập liệu, không cần chờ gửi form lên máy chủ.

- **Giảm tải cho máy chủ:** Giúp máy chủ không phải xử lý các dữ liệu không hợp lệ.

- **Bảo mật:** Ngăn chặn một số loại dữ liệu độc hại hoặc không mong muốn được gửi lên.

❖ Các bước kiểm tra dữ liệu Form bằng JavaScript

Để kiểm tra dữ liệu form, bạn cần thực hiện theo các bước sau:

1. Ngăn chặn Form gửi đi mặc định

Khi người dùng nhấn nút `submit`, form sẽ tự động gửi dữ liệu và tải lại trang. Để có thể kiểm tra dữ liệu trước, bạn cần ngăn chặn hành vi mặc định này.

- Cách thực hiện: Lắng nghe sự kiện `submit` của form và sử dụng phương thức `event.preventDefault()`.

```
const form = document.querySelector('form');
form.addEventListener('submit', function(event) {
  event.preventDefault(); // Ngăn chặn form gửi đi
  // Sau đó thực hiện kiểm tra dữ liệu
});
```

2. Truy cập và lấy giá trị các trường nhập liệu

Sử dụng DOM để truy cập các trường `<input>`, `<select>`, `<textarea>` và lấy giá trị của chúng thông qua thuộc tính `.value`.

```
const emailInput = document.getElementById('email');
const passwordInput = document.getElementById('password');

const email = emailInput.value;
const password = passwordInput.value;
```

3. Thực hiện kiểm tra dữ liệu

Bạn sẽ dùng các câu lệnh điều kiện `if/else` và các toán tử logic để kiểm tra tính hợp lệ của dữ liệu.

- **Kiểm tra rỗng:** `if (email === '')`

- **Kiểm tra độ dài:** `if (password.length < 6)`

- **Kiểm tra định dạng:** Sử dụng các biểu thức chính quy (regular expressions) để kiểm tra định dạng email hoặc số điện thoại.

Ví dụ:

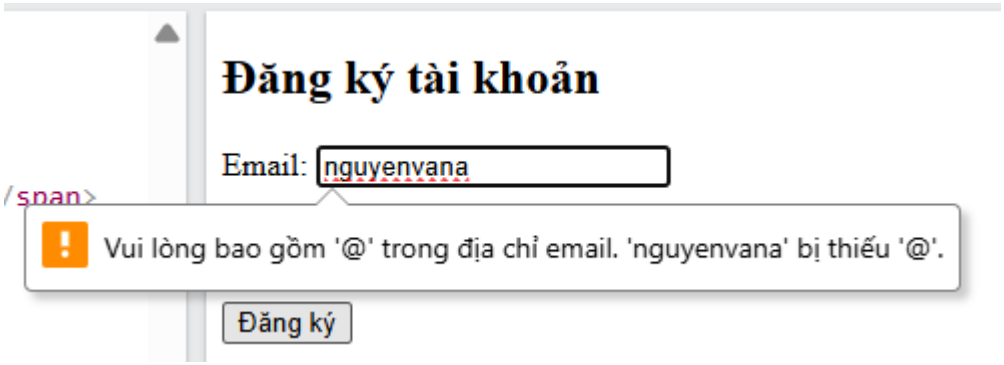
```
// Kiểm tra trường email có rỗng không
if (email === '') {
  alert('Vui lòng nhập email.');
  return; // Dừng hàm
}

// Kiểm tra mật khẩu có đủ 6 ký tự không
if (password.length < 6) {
  alert('Mật khẩu phải có ít nhất 6 ký tự.');
  return;
}
```

4. Hiển thị thông báo lỗi

Thay vì chỉ dùng `alert()`, bạn nên hiển thị thông báo lỗi ngay bên cạnh trường nhập liệu để trải nghiệm người dùng tốt hơn. Bạn có thể sử dụng JavaScript để thêm một thẻ `<span>` với thông báo lỗi khi cần.

❖ Ví dụ tổng hợp về Validation Form

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Validation Form</title>
    <style>
        .error-message {
            color: red;
            font-size: 0.9em;
            margin-top: -10px;
            margin-bottom: 10px;
            display: none; /* Ẩn thông báo lỗi ban đầu */
        }
        .show {
            display: block;
        }
    </style>
</head>
<body>

    <form id="myForm">
        <h2>Đăng ký tài khoản</h2>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <span id="email-error" class="error-message">Email không hợp lệ.</span>
        <br><br>
        <label for="password">Mật khẩu:</label>
        <input type="password" id="password" name="password">
        <span id="password-error" class="error-message">Mật khẩu phải có ít nhất 6 ký tự.</span>
        <br><br>
        <button type="submit">Đăng ký</button>
    </form>

    <script>
        const form = document.getElementById('myForm');
        const emailInput = document.getElementById('email');
        const passwordInput = document.getElementById('password');
        const emailError = document.getElementById('email-error');
        const passwordError = document.getElementById('password-error');

        form.addEventListener('submit', function(event) {
            event.preventDefault(); // Ngăn chặn submit
            // Ẩn tất cả thông báo lỗi
            emailError.classList.remove('show');
            passwordError.classList.remove('show');
            let isValid = true;
            // Kiểm tra email
            if (emailInput.value === '' || !emailInput.value.includes('@')) {
                emailError.classList.add('show');
                isValid = false;
            }
            // Kiểm tra mật khẩu
            if (passwordInput.value.length < 6) {
                passwordError.classList.add('show');
                isValid = false;
            }
            // Nếu dữ liệu hợp lệ, gửi form
            if (isValid) {
                alert('Dữ liệu hợp lệ! Đang gửi form...');
                // form.submit(); // Uncomment dòng này để gửi form thật
            }
        });
    </script>
</body>
</html>
```



Demo: Javascript/n_form_validation.html

2.3.9. Giới thiệu ES6+ (let/const, arrow function, template string…)

❖ ES6+ là gì?

- **ES6** (ECMAScript 2015), còn được gọi là **ES2015**, là một bản cập nhật lớn cho ngôn ngữ JavaScript. Sau ES6, các phiên bản mới hơn được phát hành hàng năm (ES2016, ES2017,...) và được gọi chung là **ES6+**.

- Mục tiêu chính của ES6+ là làm cho JavaScript trở nên mạnh mẽ, hiện đại và dễ đọc hơn, giúp giải quyết những vấn đề còn tồn tại ở các phiên bản cũ.

❖ Các tính năng quan trọng của ES6+

1. Khai báo biến `let` và `const`

Đây là một trong những thay đổi cơ bản và quan trọng nhất. Chúng ta đã thảo luận về chúng trong phần trước, nhưng hãy nhấn mạnh lại tầm quan trọng của chúng:

- **`let`**: Dùng để khai báo biến có thể gán lại giá trị. Nó có **phạm vi khối (block scope)**, an toàn hơn so với `var`.

- **`const`**: Dùng để khai báo hằng số, giá trị **không thể thay đổi**. Nó cũng có **phạm vi khối**. Sử dụng `let` và `const` là tiêu chuẩn hiện đại, giúp tránh các lỗi không đáng có và làm cho mã nguồn dễ dự đoán hơn.

***2. Hàm mũi tên (Arrow Functions)* =>**

Đây là một cách viết hàm ngắn gọn và hiện đại hơn.

- Cú pháp ngắn gọn hơn: Rút gọn cú pháp so với cách viết hàm truyền thống.

- Không có `this` riêng: Đây là điểm khác biệt lớn nhất. Arrow function không tự định nghĩa `this` mà sẽ kế thừa `this` từ ngữ cảnh (context) bao bọc nó.

- Ví dụ:

```
<script>
    // Cách viết hàm truyền thống
    const add = function(a, b) {
      return a + b;
    };

    // Cách viết với Arrow Function
    const addArrow = (a, b) => {
      return a + b;
    };

    // Viết ngắn gọn hơn nữa
    const addShort = (a, b) => a + b;
</script>
```

Mình hiểu rồi. Bạn muốn tìm hiểu chi tiết hơn về các tính năng chính của **ES6+** (ES2015 và các phiên bản sau này). Dưới đây là giới thiệu cụ thể về các tính năng bạn đã đề cập:

3. Template Strings (Chuỗi Mẫu)

Template strings cho phép bạn tạo chuỗi một cách dễ dàng và linh hoạt hơn bằng cách sử dụng dấu backtick `` ` ``.

Nhúng biến và biểu thức: Bạn có thể nhúng biến hoặc biểu thức JavaScript trực tiếp vào chuỗi bằng cú pháp `${...}`. Điều này loại bỏ nhu cầu phải nối chuỗi bằng toán tử `+`.

```
<script>
  const name = "Alice";
  const age = 25;
  const greeting = `Hello, my name is ${name} and I am ${age} years old.`;
  // Kết quả: "Hello, my name is Alice and I am 25 years old."
</script>
```

Hỗ trợ chuỗi đa dòng: Template strings cũng cho phép bạn viết chuỗi trên nhiều dòng mà không cần dùng ký tự xuống dòng `\n`.

```
<script>
  const multiLineString = `
    This is a string
    on multiple lines.
  `;
</script>
```

4. Classes (Lớp)

Classes là một cú pháp "đường cú pháp" (syntactic sugar) trên hệ thống kế thừa dựa trên prototype của JavaScript. Chúng giúp code dễ đọc và dễ hiểu hơn đối với những người quen thuộc với lập trình hướng đối tượng (OOP).

- Cú pháp khai báo:

```
<script>
    class Person {
      constructor(name) {
        this.name = name; // Phương thức khởi tạo
      }
      sayHello() {
        console.log(`Hello, my name is ${this.name}`);
      }
    }
    const person1 = new Person("Bob");
    person1.sayHello(); // "Hello, my name is Bob"
</script>
```

- **Kế thừa**: Classes cũng hỗ trợ kế thừa bằng từ khóa `extends` và gọi phương thức của lớp cha bằng `super()`.

```
<script>
      class Person {
        constructor(name) {
          this.name = name; // Phương thức khởi tạo
        }
        sayHello() {
          console.log(`Hello, my name is ${this.name}`);
        }
      }

      class Employee extends Person {
        constructor(name, job) {
          super(name); // Gọi constructor của lớp cha
          this.job = job;
        }
        describe() {
          console.log(`${this.name} works as a ${this.job}.`);
        }
      }
      const emp1 = new Employee("Charlie", "Developer");
      emp1.describe(); // "Charlie works as a Developer.
</script>
```

Những tính năng này đã làm cho JavaScript trở nên mạnh mẽ, hiện đại và dễ bảo trì hơn rất nhiều. Học và sử dụng chúng là một bước quan trọng để trở thành một lập trình viên JavaScript hiệu quả.

2.4. Kết hợp HTML + CSS + JavaScript

2.4.1. Tạo trang web có bố cục hoàn chỉnh

Để tạo một trang web hoàn chỉnh, bạn cần ba thành phần chính:

- HTML (HyperText Markup Language): Đóng vai trò là khung xương của trang web, xác định cấu trúc và nội dung chính như các tiêu đề, đoạn văn, hình ảnh, liên kết, và các khối nội dung.

- CSS (Cascading Style Sheets): Là lớp da và trang phục của trang web. CSS giúp bạn định hình, tạo màu sắc, font chữ và sắp xếp các thành phần HTML, biến một trang web thô sơ trở nên hấp dẫn và dễ nhìn.

- JavaScript (JS): Là bộ não của trang web, mang lại sự tương tác và tính năng động. Với JavaScript, trang web của bạn có thể phản hồi các hành động của người dùng (như nhấn nút), hiển thị dữ liệu động, hoặc thực hiện các hiệu ứng phức tạp.

❖ **Các bước thực hiện**

Bước 1: Chuẩn bị cấu trúc thư mục

Tạo một thư mục dự án và bên trong đó, tạo các tệp sau để giữ cho mã nguồn của bạn được tổ chức gọn gàng:

`index.html`: Chứa toàn bộ nội dung HTML.

`css/style.css`: Chứa tất cả các quy tắc CSS để định dạng.

`js/script.js`: Chứa mã JavaScript để thêm các chức năng tương tác.

Bước 2: Xây dựng cấu trúc HTML cơ bản (`index.html`)

Mở tệp `index.html` và viết mã HTML. Bạn sẽ định nghĩa các khối nội dung chính như tiêu đề, thanh điều hướng, nội dung chính, và chân trang.

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Trang Web của tôi</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <h1>Chào mừng đến với Trang Web của tôi</h1>
        <nav>
            <ul>
                <li><a href="#">Trang chủ</a></li>
                <li><a href="#">Giới thiệu</a></li>
                <li><a href="#">Dịch vụ</a></li>
                <li><a href="#">Liên hệ</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section class="content-section">
            <h2>Phần Nội dung Chính</h2>
            <p>Đây là một đoạn văn bản mẫu.</p>
            <button id="changeTextBtn">Nhấn vào đây</button>
        </section>
    </main>

    <footer>
        <p>&copy; 2024 Trang Web của tôi. Tất cả bản quyền thuộc về.</p>
    </footer>

    <script src="js/script.js"></script>
</body>
</html>
```

Bước 3: Định dạng CSS (`style.css`)

Trong tệp `style.css`, bạn sẽ thêm các quy tắc để làm cho trang web trông đẹp mắt. Ví dụ, bạn có thể định dạng font chữ, màu sắc, và sử dụng **Flexbox** hoặc **Grid** để tạo bố cục linh hoạt.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}

header {
    background-color: #333;
    color: #fff;
    padding: 1rem 0;
    text-align: center;
}

header nav ul {
    list-style: none;
    padding: 0;
    display: flex; /* Dùng Flexbox để sắp xếp các mục theo chiều ngang */
    justify-content: center;
}

header nav ul li {
    margin: 0 15px;
}

main {
    padding: 20px;
    display: flex;
    justify-content: center;
}

.content-section {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    max-width: 800px;
    text-align: center;
}

button {
    background-color: #007bff;
    color: #fff;
    border: none;
    padding: 10px 20px;
    cursor: pointer;
    border-radius: 5px;
    margin-top: 10px;
}
```

Bước 4: Thêm tương tác với JavaScript (`script.js`)

Cuối cùng, mở tệp `script.js` để thêm các chức năng. Bạn sẽ sử dụng JavaScript để lắng nghe một sự kiện (ví dụ: người dùng nhấp vào nút) và thực hiện một hành động (ví dụ: thay đổi nội dung).

```
// Lấy phần tử HTML có id là "changeTextBtn"
const changeTextBtn = document.getElementById('changeTextBtn');

// Lấy phần tử HTML có class là "content-section"
const contentSection = document.querySelector('.content-section');

// Thêm sự kiện 'click' cho nút
changeTextBtn.addEventListener('click', function() {
    // Thay đổi nội dung của thẻ p bên trong content-section
    const paragraph = contentSection.querySelector('p');
    paragraph.textContent = 'Bạn vừa thay đổi nội dung bằng JavaScript!';
});
```

Sau khi hoàn thành các bước trên, bạn có thể mở tệp index.html trong trình duyệt. Bạn sẽ thấy một trang web với bố cục hoàn chỉnh, được định dạng đẹp mắt và có chức năng tương tác đơn giản. Điều quan trọng là bạn đã tách bạch ba ngôn ngữ này ra các tệp riêng biệt, làm cho việc quản lý mã nguồn dễ dàng hơn rất nhiều.

Demo: webgroup/chuong2th hoặc WebDemo5

2.4.2. Giới thiệu cơ bản về AJAX

- AJAX (viết tắt của Asynchronous JavaScript and XML) là một kỹ thuật lập trình web cho phép bạn cập nhật một phần của trang web mà không cần tải lại toàn bộ trang.

- Hãy tưởng tượng một trang web như một ngôi nhà.

+ Cách truyền thống: Khi bạn muốn thay đổi một bức tranh trên tường, bạn phải phá bỏ toàn bộ ngôi nhà cũ và xây lại một ngôi nhà hoàn toàn mới, chỉ để đổi một bức tranh. Rất tốn thời gian và lãng phí.

+ Với AJAX: Bạn chỉ cần gỡ bức tranh cũ xuống và treo bức tranh mới lên. Phần còn lại của ngôi nhà (trang web) vẫn nguyên vẹn.

- Về cơ bản, AJAX cho phép trình duyệt gửi và nhận dữ liệu từ máy chủ một cách bất đồng bộ (asynchronous). Điều này có nghĩa là trang web có thể tiếp tục hoạt động bình thường (ví dụ: người dùng vẫn cuộn trang, nhấp chuột) trong khi nó đang tải dữ liệu mới từ máy chủ ở chế độ nền.

- Mặc dù tên có chứa "XML" (một định dạng dữ liệu), ngày nay, AJAX thường sử dụng định dạng JSON (JavaScript Object Notation) để trao đổi dữ liệu vì nó gọn nhẹ và dễ xử lý hơn.

Tóm lại, vai trò của AJAX là:

- **Tăng tốc độ và cải thiện trải nghiệm người dùng:** Người dùng không phải chờ đợi toàn bộ trang tải lại.

- **Hiển thị nội dung động:** Trang web có thể tự động cập nhật dữ liệu mới từ máy chủ (ví dụ: thông báo, tin tức, bình luận) mà không cần can thiệp từ người dùng.

- **Tạo các ứng dụng web tương tác cao:** Giúp xây dựng các tính năng như "Gợi ý tìm kiếm" (Search Suggestions), "Tải thêm bài viết" (Load More), hoặc "Like/Tim" mà không làm gián đoạn trải nghiệm.

2.5. Bảo mật cơ bản phía Client

Bảo mật phía client (Front-end Security) là việc đảm bảo các hoạt động diễn ra trên trình duyệt của người dùng được an toàn. Mục đích chính là ngăn chặn những kẻ tấn công lợi dụng lỗ hổng trên giao diện người dùng để chiếm đoạt dữ liệu hoặc thực hiện các hành động độc hại.

Đây là tuyến phòng thủ đầu tiên, mặc dù **không bao giờ là đủ** để thay thế cho bảo mật phía máy chủ (Server-side Security)

2.5.1. Nguyên tắc an toàn khi xử lý dữ liệu Form

Dữ liệu từ form là một trong những điểm yếu phổ biến nhất. Dưới đây là các nguyên tắc an toàn cơ bản:

- Không tin tưởng dữ liệu từ người dùng: Mọi dữ liệu người dùng gửi lên, dù đã được kiểm tra phía client bằng JavaScript, đều phải được kiểm tra lại một lần nữa ở phía máy chủ. Kẻ tấn công có thể dễ dàng bỏ qua hoặc làm giả các kiểm tra JavaScript của bạn.

- Kiểm tra tính hợp lệ (Validation):

+ Phía client: Sử dụng JavaScript để kiểm tra nhanh các trường như định dạng email, độ dài mật khẩu... Điều này giúp cải thiện trải nghiệm người dùng bằng cách cung cấp phản hồi ngay lập tức, nhưng không đảm bảo an toàn.

+ Phía máy chủ: Đây là bước bắt buộc. Sau khi dữ liệu được gửi đến máy chủ, bạn phải kiểm tra lại tất cả các dữ liệu đó một cách cẩn thận trước khi xử lý hoặc lưu trữ.

- Lọc dữ liệu (Sanitization): Loại bỏ hoặc vô hiệu hóa các ký tự độc hại (như < hay >) trong dữ liệu nhập vào để ngăn chặn các cuộc tấn công tiêm mã (Injection Attacks).

2.5.2. Giới thiệu XSS, CSRF

Đây là hai kiểu tấn công phổ biến nhất mà lập trình viên front-end cần biết.

Tấn công XSS (Cross-Site Scripting):

- Khái niệm: Kẻ tấn công tiêm các đoạn mã độc (thường là JavaScript) vào một trang web để thực thi trên trình duyệt của người dùng khác.

- Cơ chế: Kẻ tấn công gửi một nội dung chứa mã JavaScript độc hại (ví dụ: `<script>alert('Hello!');</script>`) vào một form bình luận. Khi người dùng khác truy cập trang web có bình luận đó, trình duyệt của họ sẽ thực thi đoạn mã độc này.

- Mục đích: Đánh cắp thông tin nhạy cảm như cookie, session, hoặc chuyển hướng người dùng đến một trang web lừa đảo.

Tấn công CSRF (Cross-Site Request Forgery):

- Khái niệm: Kẻ tấn công lừa người dùng đã đăng nhập vào một trang web thực hiện một hành động mà họ không hề hay biết.

- Cơ chế: Ví dụ, một kẻ tấn công gửi một email lừa đảo với một hình ảnh ẩn. Khi người dùng mở email đó, trình duyệt sẽ tự động tải hình ảnh, nhưng thực chất, đó là một yêu cầu để chuyển 1000$ từ tài khoản ngân hàng của họ. Vì người dùng đang đăng nhập, yêu cầu này được coi là hợp lệ.

- Mục đích: Thực hiện các hành động trái phép như chuyển tiền, thay đổi mật khẩu, hoặc đăng bài viết giả mạo.

2.5.3. Cookie, Session và Local Storage

Đây là những cơ chế chính để lưu trữ dữ liệu người dùng trên trình duyệt.

Cookie:

- Là gì? Một tệp văn bản nhỏ được máy chủ gửi đến trình duyệt và lưu trữ trên máy tính của người dùng.

- Mục đích: Chủ yếu để lưu trữ thông tin đăng nhập (như ID phiên) hoặc sở thích của người dùng để nhận diện họ trong các lần truy cập tiếp theo.

- Đặc điểm: Dữ liệu nhỏ, có thể hết hạn, và tự động gửi kèm theo mỗi yêu cầu HTTP đến máy chủ. Điều này khiến chúng dễ bị tấn công CSRF và XSS.

Local Storage:

- Là gì? Một bộ nhớ lớn hơn nhiều, được tích hợp sẵn trong trình duyệt để lưu trữ dữ liệu của trang web.

- Mục đích: Lưu trữ dữ liệu cục bộ như cài đặt giao diện, dữ liệu người dùng không nhạy cảm hoặc bộ nhớ cache của ứng dụng.

- Đặc điểm: Dữ liệu không tự động gửi đến máy chủ. Nó chỉ có thể được truy cập bằng JavaScript, làm giảm rủi ro bị tấn công CSRF.

Session Storage:

- Là gì? Tương tự như Local Storage nhưng chỉ lưu trữ dữ liệu cho một phiên duyệt web (cho đến khi tab hoặc cửa sổ trình duyệt bị đóng).

- Mục đích: Lưu trữ dữ liệu tạm thời cần thiết cho một phiên làm việc cụ thể.

- Đặc điểm: Dữ liệu sẽ bị xóa khi tab trình duyệt đóng, giúp đảm bảo tính riêng tư cho phiên làm việc hiện tại.

BÀI TẬP CHƯƠNG

- Xây dựng giao diện web tĩnh với HTML, CSS

- Thêm JavaScript để tương tác với người dùng.

- Thêm Form và kiểm tra nhập dữ liệu trên form

CHƯƠNG 3: LẬP TRÌNH PHÍA SERVER VỚI JSP VÀ JDBC

MỤC TIÊU:

1. Nắm khái niệm cơ bản về lập trình web với JSP; nắm cách tạo và biên dịch website JSP với Apache Tomcat; nắm cách kết nối cơ sở dữ liệu với JDBC.

2. Hiểu rõ các thẻ và chỉ thị JSP; biết cách tương tác với Form trong JSP; biết cách kết nối CSDL với JSP.

3. Xây dựng ứng dụng JSP phía server với CSDL theo mô hình 3 tầng MVC.

NỘI DUNG CHI TIẾT

3.1. Giới thiệu Lập trình phía Server với JSP

Lập trình phía Server (Server-side Programming) là phần cốt lõi của việc xây dựng các ứng dụng web phức tạp. Nếu các công nghệ phía client (HTML, CSS, JavaScript) làm nhiệm vụ hiển thị giao diện và xử lý tương tác trên trình duyệt, thì lập trình phía server lại đảm nhận các tác vụ xử lý logic, quản lý dữ liệu và tương tác với cơ sở dữ liệu.

3.1.1. Khái niệm Server-side Programming

Lập trình phía Server là việc tạo ra các chương trình, script chạy trên máy chủ (server) của một ứng dụng web. Các chương trình này chịu trách nhiệm:

- **Xử lý yêu cầu (Requests):** Tiếp nhận các yêu cầu từ trình duyệt của người dùng (ví dụ: yêu cầu đăng nhập, gửi form, tìm kiếm sản phẩm).

- **Xử lý logic nghiệp vụ:** Thực hiện các phép tính, kiểm tra, xác thực dữ liệu, và tương tác với các hệ thống backend khác.

- **Tương tác với cơ sở dữ liệu:** Thêm, sửa, xóa, hoặc truy vấn dữ liệu từ database.

- **Tạo ra phản hồi (Responses):** Trả về kết quả cho trình duyệt, thường là một trang HTML động, một tệp JSON, hoặc một tệp tin.

Khi bạn nhập một địa chỉ website hoặc nhấp vào một liên kết, trình duyệt sẽ gửi một yêu cầu (request) đến máy chủ. Máy chủ sẽ chạy các chương trình server-side để xử lý yêu cầu đó và tạo ra một phản hồi (response) để gửi ngược lại trình duyệt của bạn.

3.1.2. Vai trò của JSP trong mô hình web động

JSP (viết tắt của **JavaServer Pages**) là một công nghệ phía server của Java, được thiết kế đặc biệt để tạo ra các trang web động.

- **HTML động:** JSP cho phép bạn nhúng các đoạn mã Java (Java code) vào trong một trang HTML. Điều này giúp bạn tạo ra nội dung trang web không phải là tĩnh mà thay đổi linh hoạt dựa trên dữ liệu từ máy chủ. Ví dụ, hiển thị tên người dùng đã đăng nhập, danh sách sản phẩm từ cơ sở dữ liệu, hoặc kết quả tìm kiếm.

- **Tách biệt logic và giao diện:** JSP giúp tách biệt phần giao diện người dùng (HTML, CSS) với phần logic xử lý (Java). Điều này làm cho việc phát triển và bảo trì trở nên dễ dàng hơn. Lập trình viên thiết kế giao diện có thể tập trung vào HTML/CSS, trong khi lập trình viên Java tập trung vào logic nghiệp vụ.

3.1.3. So sánh JSP với Servlet truyền thống

JSP và Servlet đều là công nghệ của Java để tạo ra các ứng dụng web. Tuy nhiên, chúng có vai trò và cách tiếp cận khác nhau.

Đặc điểm	**JSP**	**Servlet truyền thống**
Vai trò chính	**View** (Hiển thị giao diện)	**Controller** (Xử lý logic)
Cách tiếp cận	**Code-centric HTML:** Bạn nhúng mã Java vào trong HTML.	**HTML-centric Java:** Bạn nhúng mã HTML vào trong Java (thường qua `out.println()`).
Lợi thế	Dễ dàng thiết kế giao diện, đặc biệt khi cần tạo các trang HTML phức tạp.	Phù hợp để xử lý các logic nghiệp vụ phức tạp, truy vấn cơ sở dữ liệu.
Quá trình thực thi	Server-side tự động dịch tệp `.jsp` thành một Servlet Java, biên dịch và chạy nó.	Phải tự viết toàn bộ logic trong một class Java, biên dịch và triển khai lên server.

Trong thực tế, JSP và Servlet thường được sử dụng cùng nhau theo mô hình **MVC (Model-View-Controller)**. Servlet làm nhiệm vụ xử lý yêu cầu và logic (Controller), sau đó chuyển dữ liệu cho JSP để hiển thị giao diện (View).

3.1.4. Tổng quan kiến trúc JSP (Request – Response)

Kiến trúc JSP hoạt động theo một quy trình cụ thể để xử lý các yêu cầu từ người dùng và trả về phản hồi:

1. Request (Yêu cầu): Trình duyệt của người dùng gửi một yêu cầu HTTP đến máy chủ web, trỏ đến một trang `.jsp`.

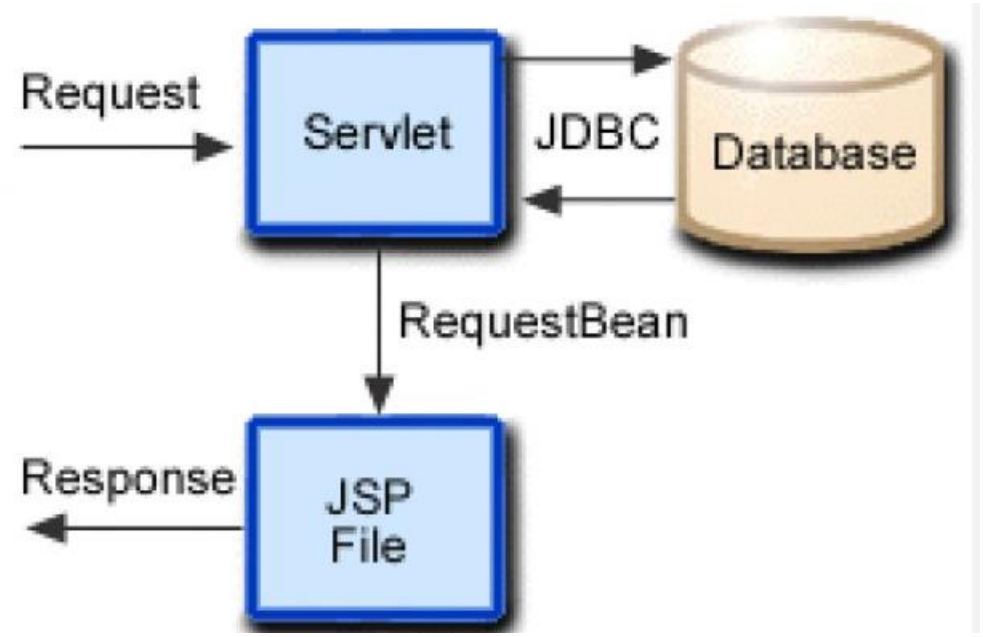
2. JSP Engine: Máy chủ web nhận yêu cầu và chuyển nó đến JSP Engine.

3. Dịch (Translation): Nếu đây là lần đầu tiên trang JSP được yêu cầu, JSP Engine sẽ dịch trang `.jsp` thành một Servlet Java.

4. Biên dịch (Compilation): Servlet Java này được biên dịch thành mã byte (`.class` file), sẵn sàng để chạy.

5. Thực thi (Execution): Servlet được thực thi. Nó có thể tương tác với cơ sở dữ liệu, xử lý dữ liệu và tạo ra một trang HTML động.

6. Response (Phản hồi): Kết quả HTML được gửi ngược lại trình duyệt. Trình duyệt nhận được trang HTML này và hiển thị nó cho người dùng, giống như một trang web tĩnh bình thường.



3.2. Môi trường phát triển JSP

Để viết và chạy các trang JSP, bạn cần một môi trường phát triển phù hợp. Điều này bao gồm một máy chủ ứng dụng web để thực thi các tệp JSP và một IDE (môi trường phát triển tích hợp) để giúp bạn viết mã một cách hiệu quả.

3.2.1. Cài đặt và cấu hình Apache Tomcat

Apache Tomcat là một máy chủ ứng dụng web mã nguồn mở, được sử dụng rộng rãi để chạy các ứng dụng Java, bao gồm cả JSP và Servlet. Nó hoạt động như một "ngôi nhà" cho ứng dụng web của bạn, tiếp nhận các yêu cầu từ trình duyệt và trả về phản hồi.

Tại sao cần Tomcat? Trình duyệt không thể tự chạy mã JSP/Java. Tomcat đóng vai trò là một servlet container, nó dịch và thực thi các trang JSP/Servlet, sau đó gửi kết quả HTML về cho trình duyệt.

Các bước cơ bản:

Các bước cài đặt có trong nội dung Hướng dẫn cài đặt các công cụ thực hành cho môn học

3.2.2. Cấu hình dự án JSP trên NetBeans (hoặc IDE khác)

- Cài NetBeans IDE: Trong Hướng dẫn cài đặt các công cụ thực hành cho môn học
- Tích hợp Tomcat với IDE: Trong Hướng dẫn cài đặt các công cụ thực hành cho môn học
- Tạo Project JSP:
  + New Project → Java Web → Web Application.
  + Đặt tên, chọn server (Tomcat), chọn Java EE version.
  + IDE tạo sẵn cấu trúc thư mục chuẩn (web pages, WEB-INF…).
- Ưu điểm: IDE tự động triển khai (deploy) ứng dụng vào Tomcat, tiết kiệm thời gian.

3.2.3. Cấu trúc thư mục ứng dụng JSP

Một ứng dụng web Java chuẩn có cấu trúc:

```
MyWebApp/
 ├─ Web Pages/ (hoặc webapp/)
 │    ├─ index.jsp
 │    ├─ other.jsp
 │    ├─ css/
 │    ├─ js/
 │    └─ images/
 ├─ WEB-INF/
 │    ├─ web.xml (Deployment descriptor)
 │    ├─ classes/ (Java classes đã biên dịch)
 │    └─ lib/ (các file .jar thư viện)
 └─ META-INF/
```

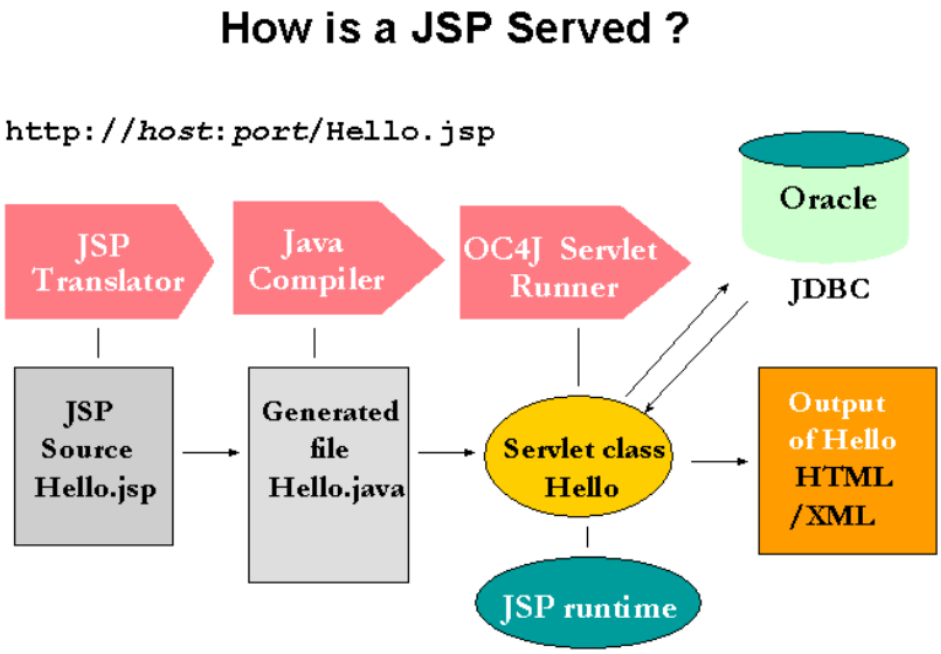
- Web pages: chứa JSP/HTML/CSS/JS.
- WEB-INF: không truy cập trực tiếp từ browser, chứa cấu hình và class.
- classes: chứa file .class sau khi biên dịch.
- lib: chứa thư viện bên ngoài (.jar).

3.2.4. Chu trình biên dịch JSP thành Servlet

- JSP thực chất không chạy trực tiếp; khi server nhận request lần đầu tới trang JSP, Tomcat:

1. Dịch JSP → Servlet Java (một file .java).
2. Biên dịch Servlet → file .class.
3. Nạp vào bộ nhớ và thực thi như Servlet bình thường.

- Lần sau khi truy cập, Tomcat chỉ chạy lại Servlet đã biên dịch (nhanh hơn).



3.3. Cấu trúc và thành phần JSP

JSP được xây dựng dựa trên một cú pháp đặc biệt, cho phép bạn kết hợp mã Java và các thẻ JSP vào trong một tệp HTML. Hiểu rõ các thành phần này là chìa khóa để tạo ra các trang web động hiệu quả.

3.3.1. Cú pháp JSP và Scriptlet

Cú pháp chính của JSP là các dấu ngoặc nhọn `<% ... %>` và `<%@ ... %>`. Những dấu này báo hiệu cho JSP Engine biết rằng đây là mã Java hoặc một chỉ thị JSP, chứ không phải là HTML thông thường.

Scriptlet (tập lệnh thực thi) là thành phần cơ bản nhất để nhúng mã Java vào trang JSP.

- Cú pháp: <% java code here %>

- Mục đích: Viết các đoạn mã logic, khai báo biến, hoặc thực thi các câu lệnh điều kiện và vòng lặp.

Ví dụ:

```
<body>
<%
    // Đây là một scriptlet
    String userName = "Học viên";
    out.println("Xin chào, " + userName + "!");
%>
</body>
```

3.3.2. Các chỉ thị (Directives): page, include, taglib

page: Cấu hình các thuộc tính trang JSP (ngôn ngữ, mã hóa ký tự, lỗi, import class).

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

include: Chèn nội dung từ file khác vào trang JSP khi biên dịch.

<%@ include file="header.jsp" %>

taglib: Khai báo thư viện thẻ tùy chỉnh sử dụng trong trang

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

3.3.3. Các phần tử JSP:

Scriptlet <% … %>: Chứa mã Java thực thi trong Servlet sinh ra.

Expression <%= … %>: In giá trị của biểu thức Java trực tiếp ra HTML.

<%= new java.util.Date() %>

Declaration <%! … %>: Khai báo biến, phương thức cho toàn bộ trang JSP.

<%! int counter = 0; %>

3.3.4. Các thẻ JSP chuẩn (Standard Actions)

Các thẻ này do JSP cung cấp sẵn, dùng để thao tác với dữ liệu hoặc điều hướng:

`<jsp:include page="header.jsp" />`: Chèn nội dung động từ file JSP khác.

`<jsp:forward page="next.jsp" />`: Chuyển tiếp request sang trang khác.

`<jsp:useBean id="obj" class="mypackage.MyBean" scope="session"/>`: Khởi tạo/nhận bean Java.

`<jsp:setProperty>` và `<jsp:getProperty>`: Thiết lập hoặc lấy giá trị thuộc tính của bean.

3.3.5. Tích hợp HTML + CSS + JavaScript vào JSP

Có thể viết trực tiếp HTML, CSS, JavaScript vào file JSP giống như trong HTML bình thường.

CSS và JS có thể đặt trong <style> hoặc <script> hoặc link file ngoài:

```
<link rel="stylesheet" href="style.css">
<script src="script.js"></script>
```

JSP cho phép xuất dữ liệu động sang HTML/JS/CSS để giao tiếp phía client:

```
<script>
   let username = "<%= session.getAttribute("username") %>";
   alert("Xin chào " + username);
</script>
```

(Demo các chỉ thị page, include: webgroup/chuong3/hello.jsp và webgroup)

3.4. Quản lý dữ liệu và tương tác Form trong JSP

3.4.1. Nhận dữ liệu từ Form HTML bằng JSP

Khái niệm: Khi người dùng nhập dữ liệu trên form HTML và nhấn Submit, dữ liệu được gửi về server theo phương thức **GET** hoặc **POST**. JSP nhận dữ liệu qua **đối tượng `request`**.

Cách làm:

- Tạo form HTML

```
<form action="xuLy.jsp" method="post">
  Tên: <input type="text" name="ten"/>
  Email: <input type="email" name="email"/>
  <input type="submit" value="Gửi"/>
</form>
```

Nhận dữ liệu ở JSP:

```
<%
  String ten = request.getParameter("ten");
  String email = request.getParameter("email");
%>
<p>Xin chào <%= ten %>, email của bạn là <%= email %></p>
```

Lưu ý:

`getParameter()` lấy một giá trị (String)

`getParameterValues()` lấy mảng giá trị (checkbox, multiple select)

(Demo: webgroup/chuong3/form.jsp và webgroup)

3.4.2. Truyền dữ liệu giữa các trang JSP

- **Qua URL** (Query String): `page2.jsp?user=Hieu` và nhận bằng `request.getParameter("user")`.

- **Qua** request.setAttribute và RequestDispatcher.forward:

```
// page1.jsp
request.setAttribute("ten", "Hieu");
RequestDispatcher rd = request.getRequestDispatcher("page2.jsp");
rd.forward(request, response);
```

```
// page2.jsp
String ten = (String)request.getAttribute("ten");
out.print("Xin chào " + ten);
```

- **Qua Session** (nếu dữ liệu cần lưu dài hơn 1 request):

```
session.setAttribute("username","Hieu");
```

và lấy ở trang khác:

```
String user = (String)session.getAttribute("username");
```

(Demo: webgroup/chuong3/page1.jsp và webgroup)

3.4.3. Quản lý Session, Cookies trong JSP

❖ **Session**

- Là cơ chế lưu trữ dữ liệu tạm thời cho từng người dùng khi họ đang tương tác với website.
- Mỗi user có một session ID riêng.
- JSP có sẵn đối tượng `session`.
- Ví dụ:

```
session.setAttribute("cartItems", cart);
// Lấy ra
List<String> cart = (List<String>)session.getAttribute("cartItems");
```

❖ **Cookies**

- Lưu dữ liệu nhỏ trên trình duyệt người dùng.
- JSP thao tác bằng `javax.servlet.http.Cookie`.
- Ví dụ:

```
Cookie c = new Cookie("username", "Hieu");
c.setMaxAge(60*60*24); //1 ngày
response.addCookie(c);
// Lấy cookies
Cookie[] cookies = request.getCookies();
for(Cookie ck: cookies){
    if("username".equals(ck.getName())){
        out.print("Xin chào " + ck.getValue());
    }
}
```

❖ **Phân biệt sơ:**

- **Session**: lưu trên server, tự động hết hạn sau thời gian inactivity.
- **Cookies**: lưu trên máy khách, tồn tại lâu hơn cho lần truy cập sau.

(Demo: webgroup/chuong3/login.jsp và webgroup)

3.4.4. Kiểm tra và xử lý dữ liệu đầu vào trên Server

- Tại sao cần: tránh lỗi, bảo mật, chống SQL Injection,....

- Cách làm cơ bản:

+ Kiểm tra dữ liệu trống/null:

```
String email = request.getParameter("email");
if(email == null || email.isEmpty()){
    out.print("Email không được để trống");
}
```

+ Kiểm tra định dạng:

```
if(!email.matches("^[A-Za-z0-9+_.-]+@(.+)$")){
    out.print("Email không hợp lệ");
}
```

+ Escape (biện pháp bảo mật) dữ liệu trước khi hiển thị lại (tránh kiểu tấn công XSS):

```
String safe = org.apache.commons.text.StringEscapeUtils.escapeHtml4(userInput);
```

Nếu thao tác với DB → sử dụng **PreparedStatement** thay cho Statement để chống SQL Injection.

(Demo: webgroup/chuong3/ validate.jsp và webgroup)

3.5. Kết nối CSDL với JDBC

3.5.1. Giới thiệu JDBC và kiến trúc

JDBC (viết tắt của Java Database Connectivity) là một API chuẩn của Java, cho phép các ứng dụng Java kết nối và tương tác với các hệ quản trị cơ sở dữ liệu quan hệ (như MySQL, Oracle, PostgreSQL, SQL Server...).

Bạn có thể hình dung JDBC như một người phiên dịch giúp Java "nói chuyện" với các CSDL khác nhau. Nhờ JDBC, bạn không cần phải viết mã riêng cho từng loại CSDL. Thay vào đó, bạn chỉ cần dùng một bộ API chung, và JDBC Driver sẽ đảm nhận việc chuyển đổi các lệnh đó thành ngôn ngữ mà CSDL cụ thể hiểu được

Kiến trúc JDBC:

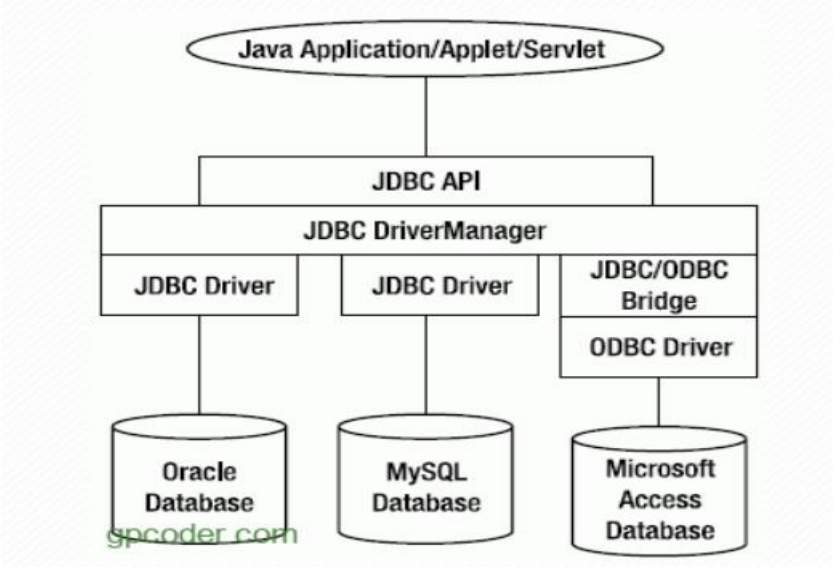
- **Java Application** (Ứng dụng Java của bạn): Nơi chứa mã Java để truy vấn CSDL.

- **JDBC API**: Giao diện lập trình chuẩn mà bạn sử dụng. Các lớp và phương thức như `Connection`, `Statement`, `ResultSet` nằm ở đây.

- **JDBC Driver Manager**: Lớp quản lý các driver, giúp tìm kiếm và tải driver phù hợp.

- **JDBC Driver**: Thư viện đặc thù cho từng loại CSDL (ví dụ: MySQL Connector/J cho MySQL). Đây chính là thành phần "phiên dịch" thực sự, chịu trách nhiệm kết nối trực tiếp với CSDL.

- **Database** (Cơ sở dữ liệu): Nơi dữ liệu của bạn được lưu tr



3.5.2. Cấu hình Driver JDBC trong dự án JSP

Để ứng dụng Java (hoặc JSP) có thể kết nối với CSDL, bạn cần phải có JDBC Driver tương ứng.

1. Tải Driver: Tải tệp `.jar` của JDBC Driver phù hợp với loại CSDL bạn đang dùng:

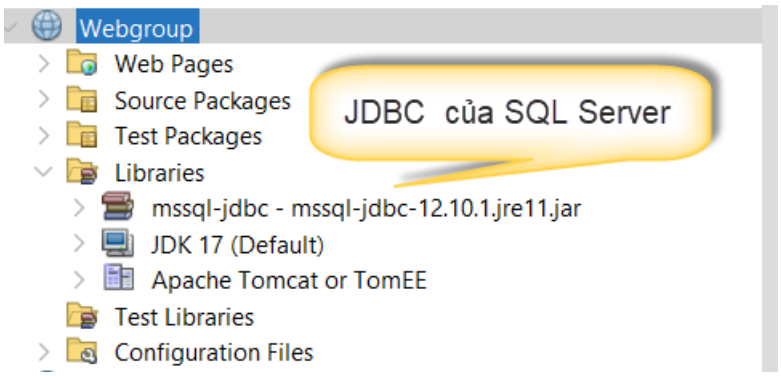
- MySQL: Tải Connector/J từ trang web chính thức của MySQL.

- SQL Server: Tải Microsoft JDBC Driver for SQL Server.

- Oracle: Tải Oracle JDBC Driver (ojdbc).

2. Thêm vào dự án: Trong các IDE như NetBeans hoặc Eclipse, bạn cần thêm tệp `.jar` này vào thư mục **`WEB-INF/lib`** của dự án web. IDE sẽ tự động cấu hình đường dẫn lớp (classpath) để ứng dụng có thể tìm thấy và sử dụng driver này.

NetBeans: Right click project → Properties → Libraries → Add JAR/Folder → Chọn driver.



3.5.3. Các bước kết nối CSDL với JDBC:

1. Load Driver

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver ");
```

2. Tạo Connection

```
String url = "jdbc:mysql://localhost:3306/quanlysv";
String user = "root";
String password = "123456";
Connection conn = DriverManager.getConnection(url, user, password);
```

3. Tạo Statement / PreparedStatement

```
Statement stmt = conn.createStatement();
PreparedStatement ps = conn.prepareStatement("SELECT * FROM sinhvien WHERE lop=?");
```

4. Thực thi Query

```
ResultSet rs = stmt.executeQuery("SELECT * FROM sinhvien");
```

5. Đọc ResultSet

```
while(rs.next()){
    int id = rs.getInt("id");
    String name = rs.getString("hoten");
    out.println(id + "-" + name);
}
```

6. Đóng kết nối

```
rs.close();
stmt.close();
conn.close();
```

3.5.4. Ví dụ minh họa CRUD (Create – Read – Update – Delete) với JDBC

1. Tạo cơ sở dữ liệu **qlsinhvien;** tạo bảng dữ liệu **sinhvien** và cập nhật thông tin cho bảng; tạo tài khoản **qlsv** trong SQL Server để quản lý cơ sở dữ liệu.

Bảng dữ liệu: **sinhvien**

	Column Name	Data Type	Allow Nulls
▶	id	int	☐
🔑	masv	nvarchar(50)	☐
	hodem	nvarchar(50)	☑
	ten	nvarchar(50)	☑
	lop	nvarchar(50)	☑
			☐

2. Tạo Project với cấu trúc như mục như sau

```
WebContent/
│
├── list.jsp       (Xem danh sách - Read)
├── add.jsp        (Thêm mới - Create)
├── edit.jsp       (Cập nhật - Update)
├── delete.jsp     (Xoá - Delete)
└── dbconnect.jsp  (File kết nối DB dùng chung)
```

3. Tạo dbconnect.jsp – File kết nối chung

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<%
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    String url =
"jdbc:sqlserver://localhost:1433;databaseName=qlsinhvien;encrypt=true;trustServerCertif
icate=true";
    String user = "qlsv";
    String password = "111111";
    Connection conn = DriverManager.getConnection(url,user,password);
%>
```

4. Tạo list.jsp – Hiển thị danh sách sinh viên (READ)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ include file="dbconnect.jsp" %>
<html>
<head>
    <title>Danh sách sinh viên</title>
</head>
<body>
<h2>Danh sách sinh viên</h2>
<a href="add.jsp">+ Thêm mới sinh viên</a><br><br>
<table border="1" cellpadding="5">
<tr><th>ID</th><th>Họ tên</th><th>Lớp</th><th>Thao tác</th></tr>
<%
    String sql = "SELECT * FROM sinhvien";
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(sql);
    while(rs.next()){
%>
<tr>
```

```
        <td><%=rs.getInt("id")%></td>
        <td><%=rs.getString("masv")%></td>
        <td><%=rs.getString("hodem")%></td>
        <td><%=rs.getString("ten")%></td>
        <td><%=rs.getString("lop")%></td>
        <td>
            <a href="edit.jsp?id=<%=rs.getInt("id")%>">Sửa</a> |
            <a href="delete.jsp?id=<%=rs.getInt("id")%>">Xoá</a>
        </td>
    </tr>
    <%
        }
        rs.close();
        stmt.close();
        conn.close();
    %>
    </table>
    </body>
    </html>
```

5. Tạo add.jsp – Form thêm mới sinh viên (CREATE)

```
    <%@page contentType="text/html" pageEncoding="UTF-8"%>
    <%@ include file="dbconnect.jsp" %>
    <html>
    <head><title>Thêm mới sinh viên</title></head>
    <body>
    <h2>Thêm mới sinh viên</h2>
    <form method="post" action="add.jsp">
        Mã số: <input type="text" name="masv"><br>
        Họ tên: <input type="text" name="hodem"><br>
        Tên: <input type="text" name="ten"><br>
        Lớp: <input type="text" name="lop"><br>
        <input type="submit" value="Thêm">
    </form>
    <%
        String masv = request.getParameter("masv");
        String hodem = request.getParameter("hodem");
        String ten = request.getParameter("ten");
        String lop = request.getParameter("lop");
        if(masv !=null && hodem != null && ten !=null && lop != null){
            String sqlInsert = "INSERT INTO sinhvien(masv,hodem,ten,lop)
VALUES(?,?,?,?)";
            PreparedStatement ps = conn.prepareStatement(sqlInsert);
            ps.setString(1,masv);
            ps.setString(2,hodem);
            ps.setString(3,ten);
            ps.setString(4,lop);
            ps.executeUpdate();
            ps.close();
            conn.close();
```

```
            response.sendRedirect("list.jsp");
        }
    %>
    </body>
    </html>
```

6. Tạo edit.jsp – Form cập nhật sinh viên (UPDATE)

```
    <%@page contentType="text/html" pageEncoding="UTF-8"%>
    <%@ include file="dbconnect.jsp" %>
    <html>
    <head><title>Cập nhật sinh viên</title></head>
    <body>
    <%
        String id = request.getParameter("id");
        if(id == null){
            response.sendRedirect("list.jsp");
        }
        String masv="", hodem="", ten="", lop="";
        // lấy dữ liệu hiện tại
        PreparedStatement psSel = conn.prepareStatement("SELECT * FROM sinhvien WHERE
id=?");
        psSel.setInt(1,Integer.parseInt(id));
        ResultSet rs = psSel.executeQuery();
        if(rs.next()){
            masv = rs.getString("masv");
            hodem = rs.getString("hodem");
            ten = rs.getString("ten");
            lop = rs.getString("lop");
        }
        rs.close();
        psSel.close();
    %>
    <h2>Cập nhật sinh viên</h2>
    <form method="post" action="edit.jsp?id=<%=id%>">
        Ma số: <input type="text" name="masv" value="<%=masv%>"><br>
        Họ tên: <input type="text" name="hodem" value="<%=hodem%>"><br>
        Tên: <input type="text" name="ten" value="<%=ten%>"><br>
        Lớp: <input type="text" name="lop" value="<%=lop%>"><br>
        <input type="submit" value="Cập nhật">
    </form>
    <%
        String newMasv = request.getParameter("masv");
        String newHodem = request.getParameter("hodem");
        String newTen = request.getParameter("ten");
        String newLop = request.getParameter("lop");
        if(newMasv != null && newHodem != null && newTen != null && newLop != null){
            PreparedStatement psUp = conn.prepareStatement("UPDATE sinhvien SET
masv=?, hodem=?, ten=?, lop=? WHERE id=?");
            psUp.setString(1,newMasv);
            psUp.setString(2,newHodem);
```

```
            psUp.setString(3,newTen);
            psUp.setString(4,newLop);
            psUp.setInt(5,Integer.parseInt(id));
            psUp.executeUpdate();
            psUp.close();
            conn.close();
            response.sendRedirect("list.jsp");
        }
    %>
    </body>
    </html>
```

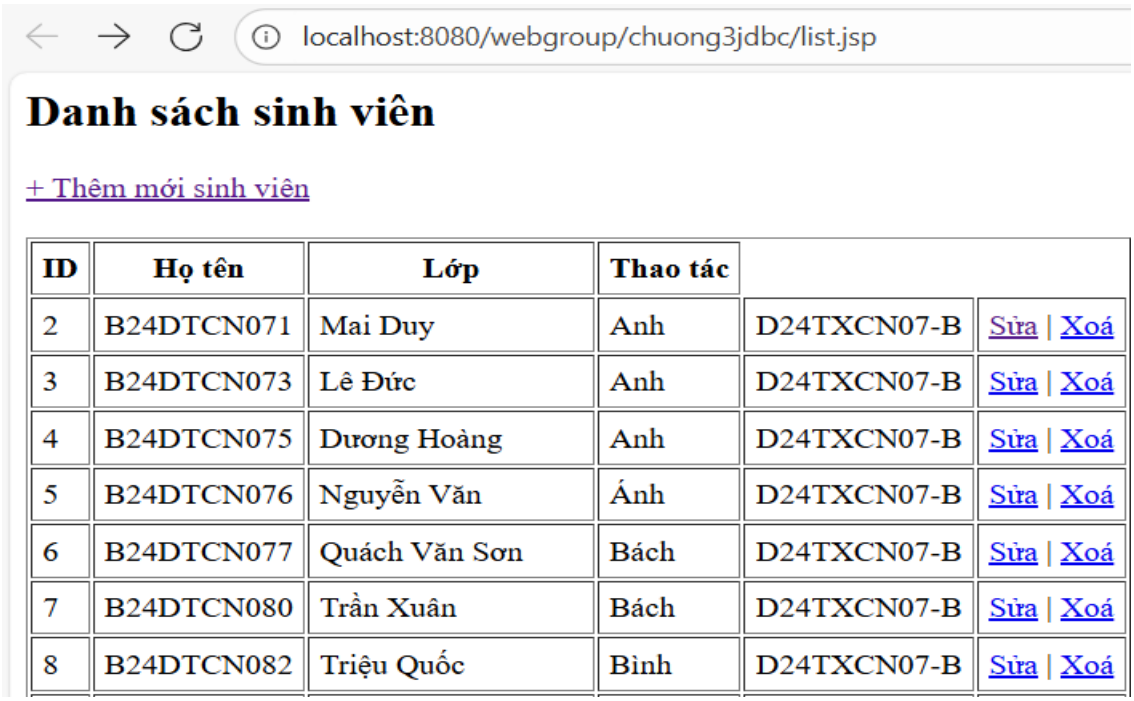
7. Tạo delete.jsp – Xoá sinh viên (DELETE)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ include file="dbconnect.jsp" %>
<%
    String id = request.getParameter("id");
    if(id != null){
        PreparedStatement psDel = conn.prepareStatement("DELETE FROM sinhvien
WHERE id=?");
        psDel.setInt(1,Integer.parseInt(id));
        psDel.executeUpdate();
        psDel.close();
    }
    conn.close();
    response.sendRedirect("list.jsp");
%>
```

(Demo: webgroup/ chuong3jdbc / list.jsp và webgroup)

❖ Kết quả thực hiện chương trình

Kết quả xem danh sách sinh viên như sau:



Kết quả cho sửa danh sách sinh viên đã có như sau:

Kết quả cho thêm sinh viên vào danh sách như sau:

3.6. Mô hình 3 Tầng MVC với JSP

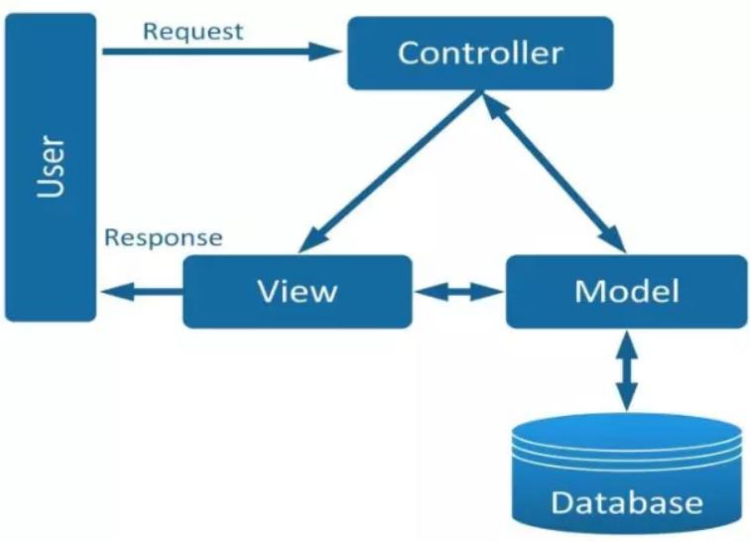
3.6.1. Khái niệm và lợi ích mô hình MVC

❖ **Khái niệm**

Mô hình MVC (Model-View-Controller) là một mẫu thiết kế kiến trúc phần mềm phổ biến, được sử dụng để tách biệt logic của ứng dụng thành ba thành phần chính. Việc này giúp việc phát triển, bảo trì và mở rộng ứng dụng trở nên dễ dàng hơn.

MVC (Model – View – Controller) là mô hình kiến trúc phần mềm giúp chia ứng dụng web thành 3 phần riêng biệt:

- Model: Quản lý dữ liệu, nghiệp vụ.
- View: Hiển thị giao diện cho người dùng.
- Controller: Điều phối luồng xử lý giữa Model và View.



❖ **Lợi ích**

- Phân tách rõ ràng giữa giao diện, xử lý dữ liệu, và điều hướng.
- Dễ bảo trì, nâng cấp, mở rộng.
- Cho phép nhiều người làm việc song song (dev backend – frontend – nghiệp vụ).
- Tái sử dụng mã (reuse code) tốt hơn.

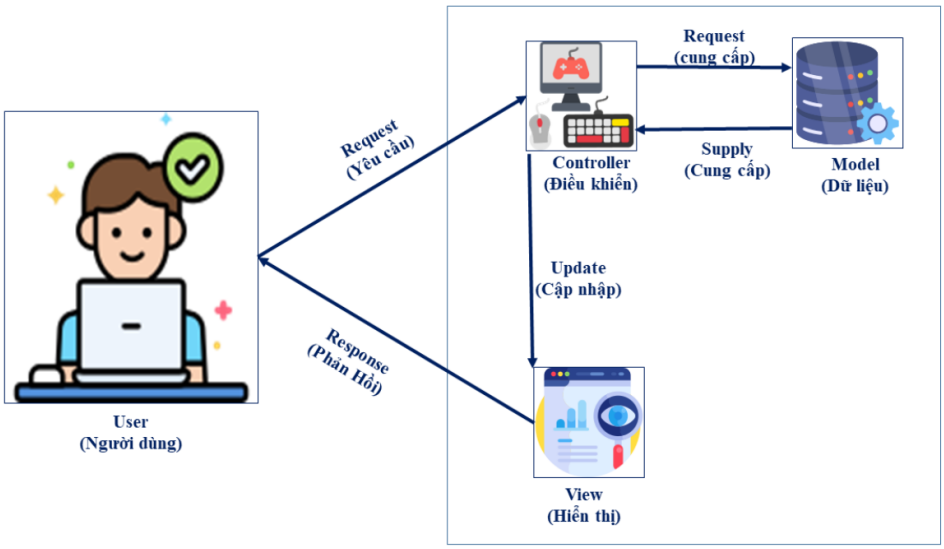
3.6.2. Phân chia các tầng

Tầng	Vai trò	Ví dụ
Model (JavaBean/DAO)	- JavaBean là các class Java đơn giản dùng để đại diện cho dữ liệu - DAO (Data Access Object): lớp truy xuất cơ sở dữ liệu, thực thi CRUD (Create – Read – Update – Delete)	`Student.java` (JavaBean): chứa các thuộc tính id, name, age) `StudentDAO.java`
View (JSP)	JSP (JavaServer Pages) được sử dụng để tạo giao diện người dùng, trình bày dữ liệu từ Model. View chỉ có nhiệm vụ hiển thị, không chứa logic xử lý nghiệp vụ: - Hiển thị dữ liệu từ Model cho người dùng. - Nhận dữ liệu nhập từ form, gửi về Controller.	`student-list.jsp`, `student-form.jsp`
Controller (Servlet)	- Nhận request từ client. - Gọi Model xử lý dữ liệu. - Điều hướng sang View thích hợp.	`StudentController.java`

3.6.3. Quy trình xử lý yêu cầu theo MVC

1. Người dùng gửi yêu cầu (request) từ trình duyệt (ví dụ: nhấn nút “Thêm sinh viên”).
2. Controller (Servlet) nhận request, đọc các tham số từ form.
3. Controller phân tích và gọi các phương thức tương ứng trong Model (JavaBean/DAO) để thực hiện logic (thêm/sửa/xóa/tìm kiếm).
4. Model xử lý dữ liệu, trả kết quả về Controller.
5. Controller quyết định chuyển hướng đến View (JSP) phù hợp để hiển thị kết quả.

6. JSP nhận dữ liệu và hiển thị giao diện người dùng tương ứng trên trình duyệt web.



3.6.4. Demo ứng dụng JSP MVC đơn giản

1. Chuẩn bị CSDL

Tạo cơ sở dữ liệu **qlsinhvien;** tạo bảng dữ liệu **sinhvien** và cập nhật thông tin cho bảng; tạo tài khoản **qlsv** trong SQL Server để quản lý cơ sở dữ liệu.

Bảng dữ liệu: **sinhvien**

Column Name	Data Type	Allow Nulls
id	int	☐
masv	nvarchar(50)	☐
hodem	nvarchar(50)	☑
ten	nvarchar(50)	☑
lop	nvarchar(50)	☑
		☐

2. Cấu trúc dự án

```
QLSinhVienMVC/
 ├─ src/
 │   ├─ model/
 │   │   ├─ SinhVien.java
 │   │   └─ SinhVienDAO.java
 │   └─ controller/
 │       └─ SinhVienController.java
 ├─ WebContent/
 │   ├─ student-list.jsp
 │   ├─ student-form.jsp
 │   ├─ student-edit.jsp
 │   └─ WEB-INF/web.xml
```

3. Thiết kế các thành phần trong cấu trúc

❖ Model:

SinhVien.java (JavaBean)

```
package model;

public class SinhVien {
    private String masv;
    private String hodem;
    private String ten;
    private String lop;

    public SinhVien() {}

    public SinhVien(String masv, String hodem, String ten, String lop) {
        this.masv = masv;
        this.hodem = hodem;
        this.ten = ten;
        this.lop = lop;
    }

    public String getMasv() { return masv; }
    public void setMasv(String masv) { this.masv = masv; }

    public String getHodem() { return hodem; }
    public void setHodem(String hodem) { this.hodem = hodem; }

    public String getTen() { return ten; }
    public void setTen(String ten) { this.ten = ten; }

    public String getLop() { return lop; }
    public void setLop(String lop) { this.lop = lop; }
}
```

SinhVienDAO.java (JDBC + CRUD)

```
package model;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SinhVienDAO {

    private static final String jdbcURL =
"jdbc:sqlserver://localhost:1433;databaseName=qlsinhvien;encrypt=true;trustServerCertificate=true";
    private static final String jdbcUsername = "qlsv";
    private static final String jdbcPassword = "111111";

    private static final String INSERT_SQL =
        "INSERT INTO sinhvien (masv, hodem, ten, lop) VALUES (?, ?, ?, ?)";
    private static final String SELECT_ALL =
        "SELECT * FROM sinhvien";
    private static final String SELECT_BY_ID =
        "SELECT * FROM sinhvien WHERE masv=?";
    private static final String UPDATE_SQL =
        "UPDATE sinhvien SET hodem=?, ten=?, lop=? WHERE masv=?";
    private static final String DELETE_SQL =
        "DELETE FROM sinhvien WHERE masv=?";
    //Kết nối dữ liêu
    public Connection getConnection() {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            return DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);
        } catch (ClassNotFoundException | SQLException e) {
```

```
            System.err.println("Lỗi: " + e.getMessage());
        }
        return null;
    }
    //Bổ sung dữ liệu
    public void insert(SinhVien sv) {
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(INSERT_SQL)) {
            ps.setString(1, sv.getMasv());
            ps.setString(2, sv.getHodem());
            ps.setString(3, sv.getTen());
            ps.setString(4, sv.getLop());
            ps.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Lỗi: " + e.getMessage());
        }
    }
    //Liệt kê dữ liệu
    public List<SinhVien> getAll() {
        List<SinhVien> list = new ArrayList<>();
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_ALL)) {
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                list.add(new SinhVien(
                    rs.getString("masv"),
                    rs.getString("hodem"),
                    rs.getString("ten"),
                    rs.getString("lop")
                ));
            }
        } catch (SQLException e) {
            System.err.println("Lỗi: " + e.getMessage());
        }
        return list;
    }
    //Tìm kiếm thông tin theo mã
    public SinhVien getByMasv(String masv) {
        SinhVien sv = null;
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_BY_ID)) {
            ps.setString(1, masv);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                sv = new SinhVien(
                    rs.getString("masv"),
                    rs.getString("hodem"),
                    rs.getString("ten"),
                    rs.getString("lop")
                );
            }
        } catch (SQLException e) {
            System.err.println("Lỗi: " + e.getMessage());
        }
        return sv;
    }
    //Cập nhật khi sửa dữ liệu
    public void update(SinhVien sv) {
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(UPDATE_SQL)) {
            ps.setString(1, sv.getHodem());
            ps.setString(2, sv.getTen());
            ps.setString(3, sv.getLop());
            ps.setString(4, sv.getMasv());
            ps.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Lỗi: " + e.getMessage());
        }
```

```
    }
    //Xóa dữ liệu
    public void delete(String masv) {
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(DELETE_SQL)) {
            ps.setString(1, masv);
            ps.executeUpdate();
        } catch (SQLException e) {
            System.err.println("Lỗi: " + e.getMessage());
        }
    }
}
```

❖ Controller

SinhVienController.java (Servlet)

```
package controller;

import model.SinhVien;
import model.SinhVienDAO;
import jakarta.servlet.*;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.*;
import java.io.IOException;
import java.util.List;

@WebServlet("/sinhvien")
public class SinhVienController extends HttpServlet {
    private SinhVienDAO dao = new SinhVienDAO();

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String action = request.getParameter("action");
        if (action == null) action = "list";

        switch (action) {
            case "new":
                request.getRequestDispatcher("student-form.jsp").forward(request, response);
                break;
            case "edit":
                String masv = request.getParameter("masv");
                SinhVien sv = dao.getByMasv(masv);
                request.setAttribute("sv", sv);
                request.getRequestDispatcher("student-edit.jsp").forward(request, response);
                break;
            case "delete":
                dao.delete(request.getParameter("masv"));
                response.sendRedirect("sinhvien?action=list");
                break;
            default: // list
                List<SinhVien> list = dao.getAll();
                request.setAttribute("list", list);
                request.getRequestDispatcher("student-list.jsp").forward(request, response);
        }
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String action = request.getParameter("action");
        if ("insert".equals(action)) {
            SinhVien sv = new SinhVien(
                request.getParameter("masv"),
                request.getParameter("hodem"),
                request.getParameter("ten"),
                request.getParameter("lop")
            );
```

```
            dao.insert(sv);
            response.sendRedirect("sinhvien?action=list");
        } else if ("update".equals(action)) {
            SinhVien sv = new SinhVien(
                request.getParameter("masv"),
                request.getParameter("hodem"),
                request.getParameter("ten"),
                request.getParameter("lop")
            );
            dao.update(sv);
            response.sendRedirect("sinhvien?action=list");
        }
    }
}
```

❖ View

student-list.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.util.List" %>
<%@ page import="model.SinhVien" %>
<%
    List<SinhVien> list = (List<SinhVien>) request.getAttribute("list");
%>
<h2>Danh sách sinh viên</h2>
<a href="sinhvien?action=new">Thêm sinh viên</a>
<table border="1">
<tr><th>Mã SV</th><th>Họ đệm</th><th>Tên</th><th>Lớp</th><th>Hành động</th></tr>
<% for(SinhVien sv : list){ %>
<tr>
<td><%=sv.getMasv()%></td>
<td><%=sv.getHodem()%></td>
<td><%=sv.getTen()%></td>
<td><%=sv.getLop()%></td>
<td>
<a href="sinhvien?action=edit&masv=<%=sv.getMasv()%>">Sửa</a> |
<a href="sinhvien?action=delete&masv=<%=sv.getMasv()%>"
   onclick="return confirm('Xóa sinh viên này?')">Xóa</a>
</td>
</tr>
<% } %>
</table>
```

student-form.jsp (Thêm sinh viên)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<h2>Thêm Sinh Viên</h2>
<form action="sinhvien" method="post">
    <input type="hidden" name="action" value="insert"/>
    Mã SV: <input type="text" name="masv"/><br/>
    Họ đệm: <input type="text" name="hodem"/><br/>
    Tên: <input type="text" name="ten"/><br/>
    Lớp: <input type="text" name="lop"/><br/>
    <input type="submit" value="Lưu"/>
</form>
```

student-edit.jsp (Cập nhật sinh viên)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="model.SinhVien" %>
<%
    SinhVien sv = (SinhVien) request.getAttribute("sv");
%>
<h2>Cập nhật Sinh Viên</h2>
<form action="sinhvien" method="post">
    <input type="hidden" name="action" value="update"/>
    Mã SV: <input type="text" name="masv" value="<%=sv.getMasv()%>" readonly/><br/>
```

```
    Họ đệm: <input type="text" name="hodem" value="<%=sv.getHodem()%>"/><br/>
    Tên: <input type="text" name="ten" value="<%=sv.getTen()%>"/><br/>
    Lớp: <input type="text" name="lop" value="<%=sv.getLop()%>"/><br/>
    <input type="submit" value="Cập nhật"/>
</form>
```

(Demo: webgroup/webgroup và webgroup)

Kết quả thực hiện chương trình

- Hiển thị danh sách sinh viên

THỰC HÀNH: JDBC, JSP, MVC

--Danh sách sinh viên--

Thêm sinh viên

Mã SV	Họ đệm	Tên	Lớp	Hành động
B24DTCN071	Mai Duy	Anh	D24TXCN07-B	Sửa | Xóa
B24DTCN073	Lê Đức	Anh	D24TXCN07-B	Sửa | Xóa
B24DTCN075	Dương Hoàng	Anh	D24TXCN07-B	Sửa | Xóa
B24DTCN076	Nguyễn Văn	Ánh	D24TXCN07-B	Sửa | Xóa
B24DTCN077	Quách Văn Sơn	Bách	D24TXCN07-B	Sửa | Xóa
B24DTCN080	Trần Xuân	Bách	D24TXCN07-B	Sửa | Xóa
B24DTCN082	Triệu Quốc	Bình	D24TXCN07-B	Sửa | Xóa

- Thêm sinh viên

THỰC HÀNH: JDBC, JSP, MVC

--Thêm Sinh Viên--

Mã sinh viên

B24DTCN099

Họ đệm

Phạm Phương

Tên

Linh

Lớp

D24TXCN07-B

Lưu

Bỏ qua

- Sửa và cập nhật sinh viên

THỰC HÀNH: JDBC, JSP, MVC

--Cập nhật Sinh Viên--

Họ đệm:

B24DTCN073

Họ đệm

Lê Đức

Tên

Anh

Lớp:

D24TXCN07-B

Cập nhật

Bỏ qua

3.7. Bảo mật cơ bản phía Server (giới thiệu khái niệm)

Bảo mật phía server là một phần quan trọng để bảo vệ ứng dụng web khỏi các cuộc tấn công và rò rỉ dữ liệu. Các nội dung dưới đây sẽ làm rõ một số kỹ thuật bảo mật cơ bản.

3.7.1. Kiểm soát truy cập với Session/Authentication

❖ **Khái niệm**

Xác thực (Authentication) là quá trình xác minh danh tính của người dùng. Trong lập trình web, điều này thường được thực hiện thông qua session. Khi người dùng đăng nhập thành công, server sẽ tạo một session duy nhất và lưu trữ thông tin người dùng trong đó

Cụ thể:

- Authentication (Xác thực): kiểm tra danh tính người dùng (ví dụ: username/password).

- Session: lưu thông tin người dùng sau khi đăng nhập để duy trì trạng thái giữa các yêu cầu (request).

❖ **Cánh thực hiện cơ bản trong chương trình**

1.Tạo form đăng nhập (login.jsp).

2. Servlet xử lý đăng nhập và kiểm tra username/password trong CSDL.

3. Nếu đúng, lưu thông tin vào HttpSession:

```
HttpSession session = request.getSession();
session.setAttribute("user", username);
response.sendRedirect("home.jsp");
```

4. Ở các trang cần bảo vệ, kiểm tra:

```
HttpSession session = request.getSession(false);
if (session == null || session.getAttribute("user") == null) {
    response.sendRedirect("login.jsp");
}
```

5. Khi đăng xuất người dùng (logout):

```
session.invalidate();
response.sendRedirect("login.jsp");
```

3.7.2. Tránh SQL Injection với PreparedStatement

SQL Injection là một kỹ thuật tấn công phổ biến, trong đó kẻ tấn công chèn các câu lệnh SQL độc hại vào dữ liệu đầu vào (ví dụ: form đăng nhập) để truy cập hoặc sửa đổi cơ sở dữ liệu trái phép

1. Cách thông thường, nguy hiểm:

```
String sql = "SELECT * FROM users WHERE username='" + user + "' AND password='" + pass + "'";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

Nếu Hacker nhập username nhập là: ' OR '1'='1, thí câu lệnh trên sẽ trở thành

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '...'
```

Điều này sẽ cho phép đăng nhập mà không cần đúng mật khẩu.

2. Cách khắc phục

- PreparedStatement là một đối tượng trong Java JDBC được sử dụng để thực thi các câu lệnh SQL đã được biên dịch trước. Thay vì nối chuỗi, PreparedStatement sử dụng các dấu hỏi ? làm tham số và gán giá trị sau.

- Ưu điểm: PreparedStatement tự động xử lý và làm sạch dữ liệu đầu vào, coi mọi giá trị được truyền vào là dữ liệu, không phải là một phần của câu lệnh SQL, do đó ngăn chặn hiệu quả các cuộc tấn công SQL Injection.

```
String sql = "SELECT * FROM users WHERE username=? AND password=?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, user);
pstmt.setString(2, pass);
ResultSet rs = pstmt.executeQuery();
```

- Hệ quản trị SQL sẽ tự động “escape: loại bỏ kỹ ký tự đặc biệt làm gây lỗi ” dữ liệu, tránh bị tấn công (injection).

- Tăng hiệu năng do SQL được biên dịch trước.

Cần lưu ý:

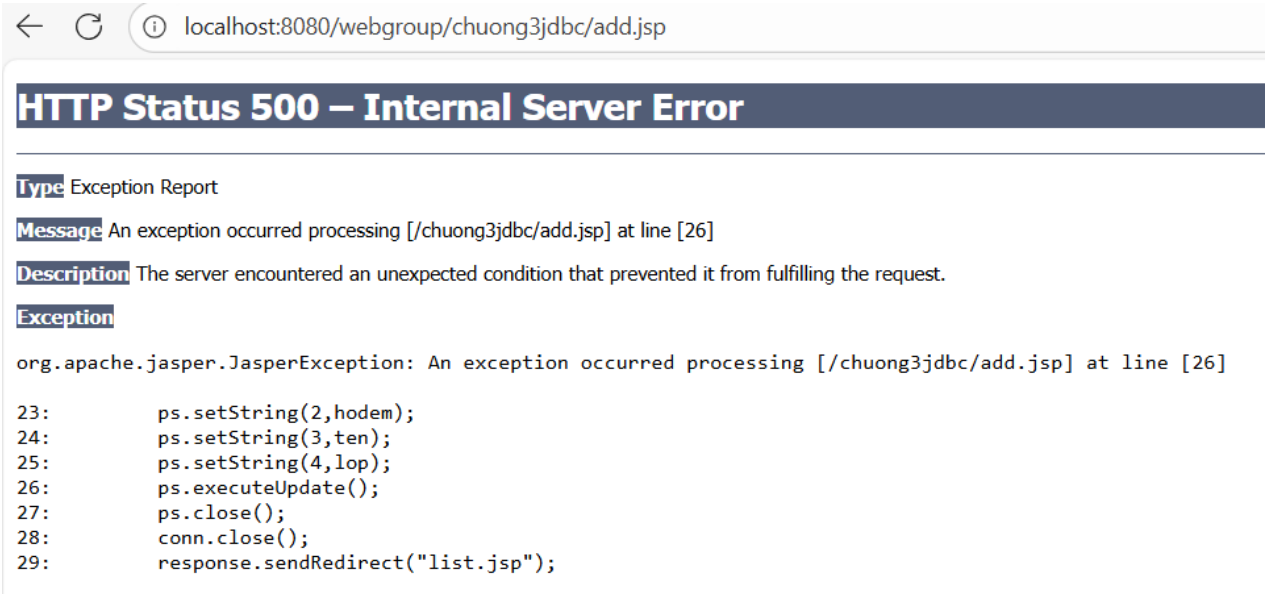
- Không bao giờ ghép chuỗi dữ liệu người dùng vào SQL trực tiếp.

- Luôn kiểm tra dữ liệu đầu vào (validation).

3.7.3. Quản lý lỗi và Exception Handling trong JSP

- Xử lý lỗi là một phần quan trọng của việc xây dựng ứng dụng web an toàn và ổn định. Một lỗi không được xử lý có thể tiết lộ thông tin nhạy cảm về ứng dụng hoặc cơ sở dữ liệu.

- Mục tiêu: Không để lỗi hiển thị “thô” ra ngoài gây lộ cấu trúc ứng dụng hoặc thông tin nhạy cảm.



❖ Cách thực hiện

1. Tạo trang lỗi riêng

Tạo trang lỗi tùy chỉnh: Thay vì để server hiển thị các trang lỗi mặc định, bạn nên tạo các trang lỗi tùy chỉnh (ví dụ: `notfound`.jsp cho lỗi không tìm thấy trang, `error`.jsp cho lỗi server nội bộ).

Trong `web.xml`, bạn có thể cấu hình để chuyển hướng người dùng đến các trang lỗi này khi xảy ra một lỗi cụ thể.

```
<error-page>
    <exception-type>500:java.lang.Exception</exception-type>
    <location>/error.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/notfound.jsp</location>
</error-page>
```

2. Trong JSP: dùng `try-catch` hoặc `errorPage`:

Trong các trang JSP

```
<%@ page errorPage="error.jsp" %>
```

Trong `error.jsp`:

```
<%@ page isErrorPage="true" %>
<h3>Đã có lỗi xảy ra: <%= exception.getMessage() %></h3>
```

3. Servlet: dùng `try-catch` và log lỗi:

```
try {
    // code
} catch (SQLException e) {
    log("Lỗi SQL", e);
    response.sendRedirect("error.jsp");
}
```

3.8. Thực hành và bài tập

- Cài đặt Tomcat, tạo project JSP đầu tiên.
- Thiết kế Form HTML gửi dữ liệu đến JSP và xử lý kết quả.
- Kết nối CSDL với JDBC để thực hiện CRUD.
- Tạo ứng dụng JSP MVC mini theo nhóm.

CHƯƠNG 4: CÁC NỀN TẢNG HỖ TRỢ PHÁT TRIỂN WEB TRÊN J2EE

MỤC TIÊU

1. Nắm được ý nghĩa của các nền tảng hỗ trợ phát triển ứng dụng web trong J2EE; Nắm được ngôn ngữ XML và cách thức xử lý file XML với Java; Hiểu được sự khác nhau giữa các nền tảng này.

2. Biết cách sử dụng một trong hai nền tảng SPRING hoặc STRUTS

3. Sử dụng thành thạo các nền tảng cho một ứng dụng web hoàn chỉnh.

NỘI DUNG CHI TIẾT

4.1. Tổng quan về các nền tảng phát triển web trên J2EE

4.1.1. Khái niệm về nền tảng (framework)

- Định nghĩa: Framework là một bộ khung (skeleton) được thiết kế sẵn, cung cấp các thư viện, công cụ và cấu trúc lập trình chuẩn để xây dựng ứng dụng web nhanh chóng, nhất quán.

- Ý nghĩa: Giúp lập trình viên không phải "làm lại từ đầu" các chức năng lặp lại như điều hướng URL, quản lý session, bảo mật, kết nối CSDL…

Ví dụ: Spring MVC, Struts, JSF trong Java; ASP.NET MVC trong .NET; Django trong Python.

4.1.2. Vai trò của framework trong phát triển ứng dụng web J2EE

- Chuẩn hóa cấu trúc dự án: Giúp dự án dễ quản lý, dễ mở rộng, giảm code rối.

- Tăng năng suất: Tận dụng các module sẵn có như Form Handling (*quá trình xử lý dữ liệu mà người dùng nhập vào từ các biểu mẫu (form) trên trang web*), Validation (*kiểm tra hợp lệ dữ liệu đầu vào*), ORM (*Object-Relational Mapping: kỹ thuật lập trình cho phép bạn tương tác với cơ sở dữ liệu bằng cách sử dụng các đối tượng (Object) của ngôn ngữ lập trình thay vì viết các câu lệnh SQL truyền thống*)

- Bảo mật & Hiệu suất: Framework thường tích hợp sẵn các giải pháp bảo mật, caching (*lưu trữ tạm thời làm giảm thời gian tải, tăng hiệu suất ứng dụng)*, tối ưu truy vấn.

- Hỗ trợ cộng đồng: Framework lớn có cộng đồng mạnh, tài liệu nhiều, cập nhật thường xuyên.

4.1.3. Lợi ích khi sử dụng framework so với phát triển thuần Servlet/JSP

Tiêu chí	Phát triển thuần Servlet/JSP	Sử dụng Framework (Spring/Struts)
Cấu trúc dự án	Tự tổ chức, dễ lộn xộn	Có sẵn mô hình MVC rõ ràng
Viết code	Nhiều code lặp lại (Session, Form, DAO)	Tận dụng thư viện có sẵn, ít code hơn
Bảo mật	Tự viết thủ công	Tích hợp sẵn Filter, Security modules
Mở rộng	Khó bảo trì khi lớn	Dễ mở rộng, module hóa rõ ràng
Học tập	Cần hiểu Servlet/JSP chi tiết	Ban đầu học thêm framework nhưng về lâu dài nhanh hơn

4.2. Ngôn ngữ XML và xử lý XML trong Java

4.2.1. Giới thiệu XML (Extensible Markup Language)

❖ **Khái niệm:**

XML (eXtensible Markup Language)là một ngôn ngữ đánh dấu, tương tự như HTML, nhưng được thiết kế để **lưu trữ và truyền tải dữ liệu**, chứ không phải để hiển thị. XML sử dụng các thẻ (tag) để mô tả cấu trúc và ý nghĩa của dữ liệu, giúp nó trở nên dễ đọc đối với cả con người và máy tính.

- Tính mở rộng (eXtensible): Bạn có thể tự định nghĩa các thẻ của riêng mình.

- Độc lập nền tảng: XML không phụ thuộc vào một ngôn ngữ lập trình hoặc hệ điều hành cụ thể nào, làm cho nó trở thành một công cụ lý tưởng để trao đổi dữ liệu giữa các hệ thống khác nhau.

Ví dụ XML sinh viên:

```
<?xml version="1.0" encoding="UTF-8"?>
<DanhSachSinhVien>
    <SinhVien>
        <MaSV>SV001</MaSV>
        <HoDem>Nguyen Van</HoDem>
        <Ten>Anh</Ten>
        <Lop>CTK43</Lop>
    </SinhVien>
    <SinhVien>
        <MaSV>SV002</MaSV>
        <HoDem>Le Thi</HoDem>
        <Ten>Bich</Ten>
        <Lop>CTK44</Lop>
    </SinhVien>
</DanhSachSinhVien>
```

❖ **Vai trò của XML trong ứng dụng web**

- Lưu cấu hình (config) cho ứng dụng (ví dụ `web.xml` trong J2EE).

- Trao đổi dữ liệu giữa client và server hoặc giữa các hệ thống (Web Services SOAP).

- Chuẩn hóa dữ liệu cho các ứng dụng tích hợp.

4.2.2. Cấu trúc file XML (elements, attributes, schema, DTD)

Một file XML có cấu trúc chặt chẽ và bao gồm các thành phần chính sau:

- **Elements (Phần tử):** Là các khối xây dựng cơ bản của XML. Mỗi phần tử được bao quanh bởi một thẻ mở (`<tên_thẻ>`) và một thẻ đóng (`</tên_thẻ>`), và có thể chứa các phần tử con khác.

Ví dụ: < `SinhVien` >, < `MaSV` >, < `HoDem` >.

- **Attributes (Thuộc tính):** Cung cấp thêm thông tin cho một phần tử. Chúng được định nghĩa bên trong thẻ mở.

Ví dụ: < `SinhVien MaSV ="S001"`>. Thuộc tính `MaSV` mang giá trị `SV001`.

- **Schema và DTD (Document Type Definition):** Đây là các cơ chế dùng để định nghĩa cấu trúc hợp lệ của một tài liệu XML. Chúng quy định các phần tử và thuộc tính nào được phép sử dụng, thứ tự và mối quan hệ giữa chúng.

+ **DTD** là một tiêu chuẩn cũ hơn và đơn giản hơn.

+ **XML Schema** là tiêu chuẩn mới hơn, mạnh mẽ và linh hoạt hơn, sử dụng chính cú pháp XML.

4.2.3. Đọc/ghi file XML trong Java: DOM, SAX, StAX

Java cung cấp nhiều API để xử lý XML. Ba API phổ biến nhất là:

- **DOM (Document Object Model):** Phân tích toàn bộ tài liệu XML và xây dựng một cây đối tượng trong bộ nhớ.

+ Ưu điểm: Dễ dàng truy cập và thay đổi các phần tử.

+ Nhược điểm: Tốn bộ nhớ khi xử lý các file XML lớn.

- **SAX (Simple API for XML):** Đọc tài liệu XML theo luồng (stream) từ đầu đến cuối và kích hoạt các sự kiện (event) khi gặp các thẻ mở, thẻ đóng, hoặc nội dung văn bản.

+ Ưu điểm: Hiệu quả về bộ nhớ, phù hợp với các file XML lớn.

+ Nhược điểm: Chỉ có thể đọc một chiều, không thể thay đổi tài liệu.

- **StAX (Streaming API for XML):** Cung cấp một mô hình dựa trên con trỏ (cursor). Lập trình viên có thể di chuyển con trỏ qua các phần tử của tài liệu.

Ưu điểm: Kết hợp được sự hiệu quả của SAX và sự linh hoạt của DOM.

Lưu ý: Khi nào chọn DOM, SAX, StAX?

- DOM: khi file nhỏ, cần thao tác linh hoạt, chỉnh sửa nhiều.

- SAX/StAX: khi file lớn, ưu tiên tốc độ và ít tốn bộ nhớ.

4.2.4. Ứng dụng XML trong cấu hình framework web

XML được sử dụng rộng rãi trong các framework web để cấu hình ứng dụng. Ví dụ điển hình là file `web.xml` (Deployment Descriptor) trong các ứng dụng Java EE (nay là Jakarta EE).

`web.xml` được sử dụng để:

- Đăng ký và ánh xạ (map) các `Servlet`.

- Định nghĩa các bộ lọc (filter).

- Cấu hình các tham số khởi tạo.

- Định nghĩa các trang lỗi tùy chỉnh.

Việc sử dụng XML cho cấu hình giúp tách biệt logic lập trình khỏi các thiết lập ứng dụng, làm cho ứng dụng trở nên linh hoạt và dễ quản lý hơn.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        version="4.0">

    <servlet>
        <servlet-name>StudentServlet</servlet-name>
        <servlet-class>com.example.controller.StudentServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>StudentServlet</servlet-name>
        <url-pattern>/students</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>AuthenticationFilter</filter-name>
        <filter-class>com.example.filter.AuthenticationFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>AuthenticationFilter</filter-name>
        <url-pattern>/admin/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <error-page>
        <error-code>404</error-code>
        <location>/error_404.jsp</location>
    </error-page>

</web-app>
```

4.2.5. Thực hành: tạo và xử lý một file XML với hành động đọc dữ liệu

- Đọc file XML vào DOM (Document)

Sơ đồ cấu trúc thư mục dự án:

```
WebAppProject/
│
├─ src/
│  ├─ model/
│  │   └─ SinhVien.java                    // Lớp model POJO
│  │
│  ├─ controller/
│  │   └─ DocSinhVienXMLServlet.java       // Servlet đọc file XML và đưa dữ liệu ra JSP
│  │
│  ├─ dao/                                 // (Tùy chọn) Nếu sau này muốn tách DAO riêng
│  │
│
├─ web/  (hoặc WebContent/)
│  ├─ WEB-INF/
│  │   ├─ web.xml                         // Cấu hình servlet (nếu cần)
│  │   └─ sinhvien.xml                    // File dữ liệu XML
│  │
│  ├─ css/
│  │   └─ style.css                       // CSS chung
│  │
│  ├─ header.jsp                          // Header chung
│  ├─ hienthiSinhVienXML.jsp              // JSP hiển thị dữ liệu từ XML
│
└─ build/ (hoặc target/)
```

1.Tạo file sinhvien.xml có cấu trúc sau, lưu tại /WEB-INF/sinhvien.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DanhSachSinhVien>
    <SinhVien>
        <masv>B24DTCN071</masv>
        <hodem>Mai Duy</hodem>
        <ten>Anh</ten>
        <lop>D24TXCN07-B</lop>
    </SinhVien>
    <SinhVien>
        <masv>B24DTCN073</masv>
        <hodem>Lê Đức</hodem>
        <ten>Anh</ten>
        <lop>D24TXCN07-B</lop>
    </SinhVien>
    <SinhVien>
        <masv>B24DTCN084</masv>
        <hodem>Hoàng Minh</hodem>
        <ten>Chiến</ten>
        <lop>D24TXCN07-B</lop>
    </SinhVien>
</DanhSachSinhVien>
```

2. Lớp Model SinhVien.java

```
package model;

public class SinhVien {
    private String masv;
    private String hodem;
    private String ten;
    private String lop;

    public SinhVien() {}

    public SinhVien(String masv, String hodem, String ten, String lop) {
        this.masv = masv;
        this.hodem = hodem;
        this.ten = ten;
        this.lop = lop;
    }

    public String getMasv() { return masv; }
    public void setMasv(String masv) { this.masv = masv; }

    public String getHodem() { return hodem; }
    public void setHodem(String hodem) { this.hodem = hodem; }

    public String getTen() { return ten; }
    public void setTen(String ten) { this.ten = ten; }

    public String getLop() { return lop; }
    public void setLop(String lop) { this.lop = lop; }
}
```

3. Servlet đọc file XML bằng DOM – DocSinhVienXMLServlet.java

```
import model.SinhVien;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.*;

@WebServlet("/docxml")
public class DocSinhVienXMLServlet extends HttpServlet {
    @Override
```

```
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            List<SinhVien> dsSV = new ArrayList<>();
            try {
                // Đường dẫn thực tế đến file XML trong /WEB-INF
                String xmlFile = getServletContext().getRealPath("/WEB-INF/sinhvien.xml");

                File inputFile = new File(xmlFile);
                DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
                DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                Document doc = dBuilder.parse(inputFile);
                doc.getDocumentElement().normalize();

                NodeList nList = doc.getElementsByTagName("SinhVien");

                for (int i = 0; i < nList.getLength(); i++) {
                    Node nNode = nList.item(i);
                    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                        Element eElement = (Element) nNode;

                        String masv =
eElement.getElementsByTagName("masv").item(0).getTextContent();
                        String hodem =
eElement.getElementsByTagName("hodem").item(0).getTextContent();
                        String ten =
eElement.getElementsByTagName("ten").item(0).getTextContent();
                        String lop =
eElement.getElementsByTagName("lop").item(0).getTextContent();

                        dsSV.add(new SinhVien(masv, hodem, ten, lop));
                    }
                }

            } catch (Exception e) {
                e.printStackTrace();
            }
            // Đưa danh sách sinh viên lên request
            request.setAttribute("dsSinhVien", dsSV);
            // Chuyển đến trang JSP để hiển thị
            RequestDispatcher rd = request.getRequestDispatcher("/hienthiSinhVienXML.jsp");
            rd.forward(request, response);
        }
    }
```

4. JSP hiển thị dữ liệu – hienthiSinhVienXML.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="model.SinhVien" %>
<link rel="stylesheet" href="css/style.css">
<jsp:include page="header.jsp" />
```

```
<html>
<head>
    <title>Danh sách Sinh Viên từ XML</title>
    <style>
        table { border-collapse: collapse; width: 70%; margin: auto; }
        th, td { border: 1px solid #999; padding: 8px; text-align: center; }
        th { background-color: #f2f2f2; }
        h2 { text-align: center; }
    </style>
</head>
<body>
<h2>Danh sách Sinh Viên từ File XML</h2>
<table>
    <tr>
        <th>Mã số</th>
        <th>Họ đệm</th>
        <th>Tên</th>
        <th>Lớp</th>
    </tr>
    <%
        List<SinhVien> ds = (List<SinhVien>) request.getAttribute("dsSinhVien");
        if (ds != null) {
            for (SinhVien sv : ds) {
    %>
    <tr>
        <td><%= sv.getMasv() %></td>
        <td><%= sv.getHodem() %></td>
        <td><%= sv.getTen() %></td>
        <td><%= sv.getLop() %></td>
    </tr>
    <%      }
        }
    %>
</table>
</body>
</html>
```

(Demo: webgroup/docxml và menu chương 4)

Kết quả thực hiện chương trình:

Danh sách Sinh Viên từ File XML

Mã số	Họ đệm	Tên	Lớp
B24DTCN071	Mai Duy	Anh	D24TXCN07-B
B24DTCN073	Lê Đức	Anh	D24TXCN07-B
B24DTCN084	Hoàng Minh	Chiến	D24TXCN07-B

4.2.6. Thực hành: tạo và xử lý một file XML với đọc, thêm, sửa, xóa

- Đọc file XML vào DOM (Document)
- Thao tác DOM (thêm node, sửa node, xóa node)
- Ghi DOM trở lại file XML (Transformer)

Sơ đồ cấu trúc thư mục dự án:

```
WebAppProject/
|
├─ src/
|  ├─ model/
|  |   └─ SinhVien.java               // Model - POJO
|  |
|  ├─ dao/
|  |   └─ XMLSinhVienDAO.java         // DAO đọc/ghi XML với DOM
|  |
|  ├─ controller/
|  |   └─ SinhVienXMLServlet.java     // Servlet xử lý request
|
├─ web/ (hoặc WebContent/)
|  ├─ WEB-INF/
|  |   ├─ web.xml                     // Cấu hình servlet (nếu cần)
|  |   └─ sinhvien.xml                // File dữ liệu XML
|  |
|  ├─ css/
|  |   └─ style.css                   // CSS chung
|  |
|  ├─ header.jsp                      // Header chung
|  ├─ xml_sinhvien_list.jsp           // JSP hiển thị danh sách
|  ├─ xml_sinhvien_add.jsp            // JSP thêm mới sinh viên
|  ├─ xml_sinhvien_edit.jsp           // JSP chỉnh sửa sinh viên
|
└─ build/ (hoặc target/)
```

1.Tạo file sinhvien.xml có cấu trúc sau, lưu tại /WEB-INF/sinhvien.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<DanhSachSinhVien>
    <SinhVien>
        <masv>B24DTCN071</masv>
        <hodem>Mai Duy</hodem>
        <ten>Anh</ten>
        <lop>D24TXCN07-B</lop>
    </SinhVien>
    <SinhVien>
        <masv>B24DTCN073</masv>
        <hodem>Lê Đức</hodem>
        <ten>Anh</ten>
        <lop>D24TXCN07-B</lop>
    </SinhVien>
    <SinhVien>
        <masv>B24DTCN084</masv>
        <hodem>Hoàng Minh</hodem>
        <ten>Chiến</ten>
```

```
        <lop>D24TXCN07-B</lop>
    </SinhVien>
</DanhSachSinhVien>
```

2. Lớp Model SinhVien.java

```
package model;

public class SinhVien {
    private String masv;
    private String hodem;
    private String ten;
    private String lop;

    public SinhVien() {}

    public SinhVien(String masv, String hodem, String ten, String lop) {
        this.masv = masv;
        this.hodem = hodem;
        this.ten = ten;
        this.lop = lop;
    }

    public String getMasv() { return masv; }
    public void setMasv(String masv) { this.masv = masv; }

    public String getHodem() { return hodem; }
    public void setHodem(String hodem) { this.hodem = hodem; }

    public String getTen() { return ten; }
    public void setTen(String ten) { this.ten = ten; }

    public String getLop() { return lop; }
    public void setLop(String lop) { this.lop = lop; }
}
```

3. DAO – Lớp `XMLSinhVienDAO.java`

```
package dao;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import model.SinhVien;
```

```
        public class XMLSinhVienDAO {
            private String xmlPath;

            public XMLSinhVienDAO(String xmlPath) {
                this.xmlPath = xmlPath;
            }
            // Đọc file XML thành Document
            private Document getDocument() throws Exception {
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();
                return builder.parse(new File(xmlPath));
            }
            // Lưu Document xuống file XML
            private void saveDocument(Document doc) throws Exception {
                TransformerFactory tf = TransformerFactory.newInstance();
                Transformer transformer = tf.newTransformer();
                transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
                transformer.setOutputProperty(OutputKeys.INDENT, "yes");
                transformer.transform(new DOMSource(doc), new StreamResult(new
File(xmlPath)));
            }
            //Thêm sinh viên
            public void addSinhVien(String masv, String hodem, String ten, String lop) throws
Exception {
                Document doc = getDocument();
                Element root = (Element)
doc.getElementsByTagName("DanhSachSinhVien").item(0);

                Element sv = doc.createElement("SinhVien");

                Element eMasv = doc.createElement("masv");
                eMasv.setTextContent(masv);
                sv.appendChild(eMasv);

                Element eHodem = doc.createElement("hodem");
                eHodem.setTextContent(hodem);
                sv.appendChild(eHodem);

                Element eTen = doc.createElement("ten");
                eTen.setTextContent(ten);
                sv.appendChild(eTen);

                Element eLop = doc.createElement("lop");
                eLop.setTextContent(lop);
                sv.appendChild(eLop);

                root.appendChild(sv);
                saveDocument(doc);
            }
```

```
            // Sửa sinh viên (theo masv)
            public void updateSinhVien(String masv, String hodem, String ten, String lop)
    throws Exception {
                Document doc = getDocument();
                NodeList list = doc.getElementsByTagName("SinhVien");
                for (int i = 0; i < list.getLength(); i++) {
                    Element sv = (Element) list.item(i);
                    String currentMasv =
    sv.getElementsByTagName("masv").item(0).getTextContent();
                    if (currentMasv.equals(masv)) {
                        sv.getElementsByTagName("hodem").item(0).setTextContent(hodem);
                        sv.getElementsByTagName("ten").item(0).setTextContent(ten);
                        sv.getElementsByTagName("lop").item(0).setTextContent(lop);
                        break;
                    }
                }
                saveDocument(doc);
            }
            // Xóa sinh viên (theo masv)
            public void deleteSinhVien(String masv) throws Exception {
                Document doc = getDocument();
                NodeList list = doc.getElementsByTagName("SinhVien");
                for (int i = 0; i < list.getLength(); i++) {
                    Element sv = (Element) list.item(i);
                    String currentMasv =
    sv.getElementsByTagName("masv").item(0).getTextContent();
                    if (currentMasv.equals(masv)) {
                        sv.getParentNode().removeChild(sv);
                        break;
                    }
                }
                saveDocument(doc);
            }

            //Hiển thị toàn bộ sinh viên
            public List<SinhVien> getAllSinhVien() throws Exception {
            Document doc = getDocument();
            List<SinhVien> list = new ArrayList<>();
            NodeList nodeList = doc.getElementsByTagName("SinhVien");
            for (int i = 0; i < nodeList.getLength(); i++) {
                Element sv = (Element) nodeList.item(i);
                String masv = sv.getElementsByTagName("masv").item(0).getTextContent();
                String hodem = sv.getElementsByTagName("hodem").item(0).getTextContent();
                String ten = sv.getElementsByTagName("ten").item(0).getTextContent();
                String lop = sv.getElementsByTagName("lop").item(0).getTextContent();
                list.add(new SinhVien(masv, hodem, ten, lop));
            }
            return list;
            }
```

```
        //Tìm kiếm thông tin theo mã
        public SinhVien getByMasv(String masv) {
            SinhVien sv = null;
            try {
                // Lấy Document
                Document doc = getDocument();

                NodeList nodeList = doc.getElementsByTagName("SinhVien");
                for (int i = 0; i < nodeList.getLength(); i++) {
                    Element element = (Element) nodeList.item(i);
                    String masvNode =
element.getElementsByTagName("masv").item(0).getTextContent();

                    if (masvNode.equalsIgnoreCase(masv)) {
                        String hodem =
element.getElementsByTagName("hodem").item(0).getTextContent();
                        String ten =
element.getElementsByTagName("ten").item(0).getTextContent();
                        String lop =
element.getElementsByTagName("lop").item(0).getTextContent();

                        sv = new SinhVien(masvNode, hodem, ten, lop);
                        break; // tìm thấy thì thoát
                    }
                }
            } catch (Exception e) {
                System.err.println("Lỗi: " + e.getMessage());
            }
            return sv;
        }
    }
```

4. Controller-Servlet: SinhVienXMLServlet.java

```
package controller;

import dao.XMLSinhVienDAO;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.*;
import java.io.IOException;
import model.SinhVien;

@WebServlet("/sinhvienxml")
public class SinhVienXMLServlet extends HttpServlet {
    private XMLSinhVienDAO dao;

    @Override
    public void init() throws ServletException {
        String xmlPath = getServletContext().getRealPath("/WEB-INF/sinhvien.xml");
```

```
            dao = new XMLSinhVienDAO(xmlPath);
        }


        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            try {
                request.setCharacterEncoding("UTF-8");
                String action = request.getParameter("action");
                if (action == null) action = "list";

                switch (action) {
                case "add":
                    request.getRequestDispatcher("xml_sinhvien_add.jsp").forward(request,
response);
                    break;
                case "edit":
                    String masv = request.getParameter("masv");
                    SinhVien sv = dao.getByMasv(masv);
                    request.setAttribute("sv", sv);
                    request.getRequestDispatcher("xml_sinhvien_edit.jsp").forward(request,
response);
                    break;
                case "delete":
                    dao.deleteSinhVien(request.getParameter("masv"));
                    response.sendRedirect("sinhvienxml?action=list");
                    break;
                default: // list
                    request.setAttribute("listSV", dao.getAllSinhVien());
                    RequestDispatcher rd =
request.getRequestDispatcher("xml_sinhvien_list.jsp");
                    rd.forward(request, response);
            }
            } catch (Exception e) {
                throw new ServletException(e);
            }
        }
        // Sau khi POST (thêm/sửa/xóa) xong, chuyển về doGet để hiển thị:
        @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            request.setCharacterEncoding("UTF-8");
            String action = request.getParameter("action");
            String masv = request.getParameter("masv");
            String hodem = request.getParameter("hodem");
            String ten = request.getParameter("ten");
            String lop = request.getParameter("lop");
```

```
        try {
            switch (action) {
                case "add":
                    dao.addSinhVien(masv, hodem, ten, lop);
                    break;
                case "update":
                    dao.updateSinhVien(masv, hodem, ten, lop);
                    break;
                case "delete":
                    dao.deleteSinhVien(masv);
                    break;
            }
        } catch (Exception e) {
            throw new ServletException(e);
        }
        // Gọi lại doGet để hiển thị danh sách mới
        response.sendRedirect("sinhvienxml");
    }
}
```

5. JSP: Hiển thị

xml_sinhvien_list.jsp: Hiển thị file xml trên web

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.util.List" %>
<%@ page import="model.SinhVien" %>
<link rel="stylesheet" href="css/style.css">
<jsp:include page="header.jsp" />
<style>
        .center {
        text-align: center;
        }
</style>
<%
    List<SinhVien> list = (List<SinhVien>) request.getAttribute("listSV");
%>
<h2 class="center">THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE</h2>
<h3 class="center">--Danh sách sinh viên--</h3>
<h4 class="center"><a href="sinhvienxml?action=add">Thêm sinh viên</a></h4>
<table border="1">
<tr><th>Mã SV</th><th>Họ đệm</th><th>Tên</th><th>Lớp</th><th>Hành động</th></tr>
<% for(SinhVien sv : list){ %>
<tr>
<td><%=sv.getMasv()%></td>
<td><%=sv.getHodem()%></td>
<td><%=sv.getTen()%></td>
<td><%=sv.getLop()%></td>
<td>
<a href="sinhvienxml?action=edit&masv=<%=sv.getMasv()%>">Sửa</a> |
```

```
<a href="sinhvienxml?action=delete&masv=<%=sv.getMasv()%>"
   onclick="return confirm('Xóa sinh viên này?')">Xóa</a>
</td>
</tr>
<% } %>
</table>
```

xml_sinhvien_edit.jsp: Sửa dữ liệu file xml

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="model.SinhVien" %>
<link rel="stylesheet" href="css/style.css">
<jsp:include page="header.jsp" />
<style>
        body { font-family: Arial; margin: 20px; }

        .form-container { max-width: 400px; margin: auto; padding: 20px; border: 1px solid
#ccc; border-radius: 8px; }
        .form-group { margin-bottom: 15px; }
        .form-group label { display: block; margin-bottom: 5px; font-weight: bold; }
        .form-group input { width: 100%; padding: 8px; box-sizing: border-box; border: 1px
solid #ccc; border-radius: 4px; }
        .error { color: red; font-size: 14px; margin-top: 5px; }
        button { padding: 10px 15px; background-color: #007bff; color: white; border:
none; border-radius: 4px; cursor: pointer; }
        button:hover { background-color: #0056b3; }
        .success { color: green; font-weight: bold; margin-top: 15px; }
        input[type=text], select {
            width: 100%; padding: 8px; margin: 5px 0; box-sizing: border-box;
        }
        input[type=submit] {
            background-color: #4CAF50; color: white; padding: 10px; border: none;
            cursor: pointer; width: 100%;
        }
        input[type=submit]:hover { background-color: #45a049; }
        .msg { color: green; }
        .err { color: red; }

        .center {
        text-align: center;
        }
</style>
<%
    SinhVien sv = (SinhVien) request.getAttribute("sv");
%>


<div class="form-container">
```

```
    <h2 class="center">THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE</h2>
    <h3 class="center">--Sửa sinh viên--</h3>

        <% if (request.getAttribute("error") != null) { %>
            <p class="err"><%= request.getAttribute("error") %></p>
        <% } %>

        <form action="sinhvienxml?action=update" method="post">
            <input type="hidden" name="action" value="update"/>
            <label>Mã sinh viên:</label>
            <input type="text" name="masv" value="<%= sv.getMasv() %>" readonly>

            <label>Họ đệm</label>
            <input type="text" name="hodem" value="<%= sv.getHodem() %>" required>

            <label>Tên</label>
            <input type="text" name="ten" value="<%= sv.getTen() %>" required>

            <label>Lớp:</label>
            <input type="text" name="lop" value="<%= sv.getLop() %>" required>

            <input type="submit" value="Cập nhật">
             <div class="center">
                <a href="sinhvienxml?action=list">Bỏ qua</a>
            </div>
        </form>
    </div>
```

xml_sinhvien_add.jsp: Bổ sung dữ liệu file xml

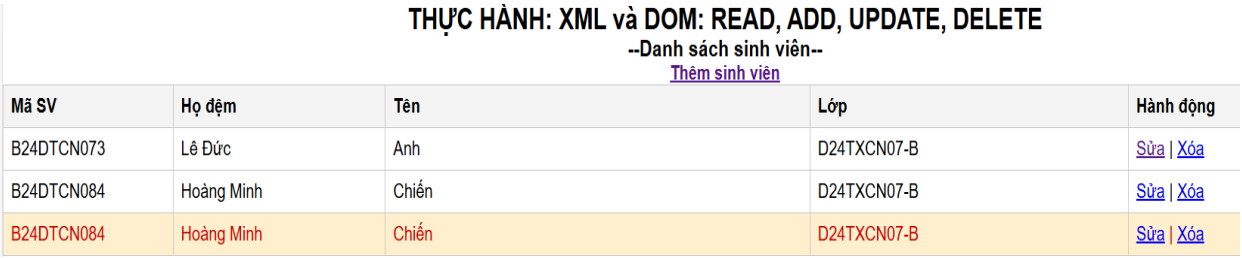
```
    <%@page contentType="text/html" pageEncoding="UTF-8"%>
    <link rel="stylesheet" href="css/style.css">
    <jsp:include page="header.jsp" />

    <style>
            body { font-family: Arial; margin: 20px; }

            .form-container { max-width: 400px; margin: auto; padding: 20px; border: 1px solid
#ccc; border-radius: 8px; }
            .form-group { margin-bottom: 15px; }
            .form-group label { display: block; margin-bottom: 5px; font-weight: bold; }
            .form-group input { width: 100%; padding: 8px; box-sizing: border-box; border: 1px
solid #ccc; border-radius: 4px; }
            .error { color: red; font-size: 14px; margin-top: 5px; }
            button { padding: 10px 15px; background-color: #007bff; color: white; border:
none; border-radius: 4px; cursor: pointer; }
            button:hover { background-color: #0056b3; }
            .success { color: green; font-weight: bold; margin-top: 15px; }
            input[type=text], select {
                width: 100%; padding: 8px; margin: 5px 0; box-sizing: border-box;
            }
```

```
        input[type=submit] {
            background-color: #4CAF50; color: white; padding: 10px; border: none;
            cursor: pointer; width: 100%;
        }
        input[type=submit]:hover { background-color: #45a049; }
        .msg { color: green; }
        .err { color: red; }

        .center {
        text-align: center;
        }
</style>
<link rel="stylesheet" href="css/style.css">


<div class="form-container">
<h2 class="center">THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE</h2>
<h3 class="center">--Thêm sinh viên--</h3>
    <form action="sinhvienxml?action=add" method="post">
        <input type="hidden" name="action" value="insert"/>
            <div class="form-group">
                <label for="masv">Mã sinh viên</label>
                <input type="text" id="masv" name="masv" required>
                <div class="error" id="nameError"></div>
            </div>
            <div class="form-group">
                <label for="hodem">Họ đệm</label>
                <input type="text" id="hodem" name="hodem" required>
                <div class="error" id="nameError"></div>
            </div>
            <div class="form-group">
                <label for="ten">Tên</label>
                <input type="text" id="ten" name="ten" required>
                <div class="error" id="nameError"></div>
            </div>
            <div class="form-group">
                <label for="lop">Lớp</label>
                <input type="text" id="lop" name="lop" required>
                <div class="error" id="nameError"></div>
            </div>

        <input type="submit" value="Lưu"/>

        <div class="center">
            <a href="sinhvienxml?action=list">Bỏ qua</a>
        </div>
    </form>
</div>
```

(Demo: webgroup/sinhvienxml và menu chương 4)

Kết quả chương trình:

- Đọc và hiển thị xml



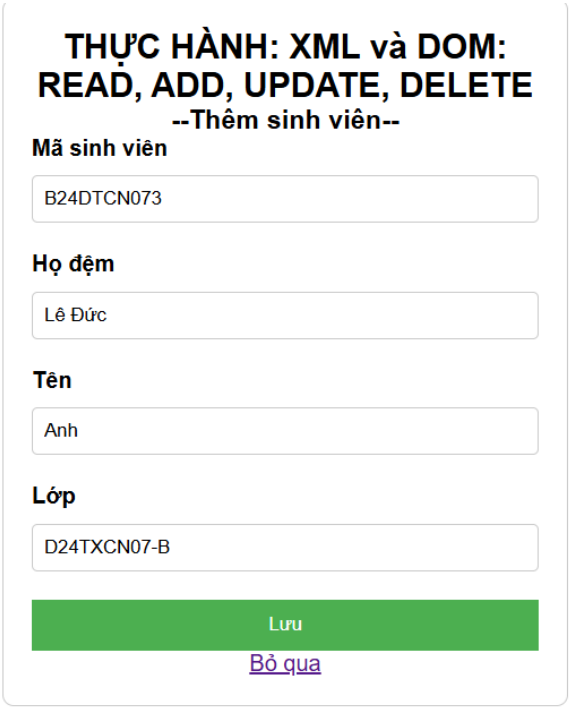
THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE

--Danh sách sinh viên--

Thêm sinh viên

Mã SV	Họ đệm	Tên	Lớp	Hành động
B24DTCN073	Lê Đức	Anh	D24TXCN07-B	Sửa | Xóa
B24DTCN084	Hoàng Minh	Chiến	D24TXCN07-B	Sửa | Xóa
B24DTCN084	Hoàng Minh	Chiến	D24TXCN07-B	Sửa | Xóa

- Bổ sung dữ liệu xml



THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE

--Thêm sinh viên--

Mã sinh viên

B24DTCN073

Họ đệm

Lê Đức

Tên

Anh

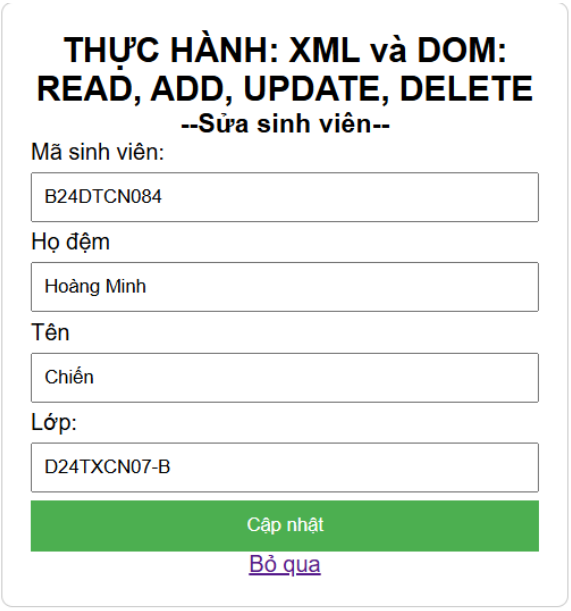
Lớp

D24TXCN07-B

Lưu

Bỏ qua

- Chỉnh sửa dữ liệu xml



THỰC HÀNH: XML và DOM: READ, ADD, UPDATE, DELETE

--Sửa sinh viên--

Mã sinh viên:

B24DTCN084

Họ đệm

Hoàng Minh

Tên

Chiến

Lớp:

D24TXCN07-B

Cập nhật

Bỏ qua

4.3. Giới thiệu các nền tảng phổ biến trên J2EE

❖ **J2EE là gì?**

- J2EE, viết tắt của **Java 2 Platform, Enterprise Edition**, là một nền tảng lập trình và một tập hợp các API (Application Programming Interfaces) của Java được thiết kế để xây dựng các ứng dụng doanh nghiệp có quy mô lớn, phân tán và đa tầng.

- J2EE ban đầu được phát triển bởi Sun Microsystems và sau đó được đổi tên thành **Java EE** (Java Platform, Enterprise Edition), nay là **Jakarta EE**. Mặc dù tên đã thay đổi, mục tiêu cốt lõi của nền tảng vẫn là cung cấp một bộ công cụ tiêu chuẩn để phát triển các ứng dụng doanh nghiệp.

❖ **Kiến trúc J2EE:**

- J2EE sử dụng kiến trúc đa tầng (multi-tier architecture), giúp tách biệt các thành phần của ứng dụng thành các lớp logic khác nhau. Điều này làm cho ứng dụng dễ quản lý, bảo trì và mở rộng.

- Một kiến trúc J2EE điển hình bao gồm bốn tầng chính:

1. Tầng Client (Client Tier-Tầng khách):

- Trình duyệt (Browser): chạy các công nghệ như HTML (HyperText Markup Language), JS (JavaScript) để hiển thị và tương tác giao diện.

- Ứng dụng Desktop hoặc Mobile: có thể là ứng dụng Java Swing, .NET, Android…

- Gửi yêu cầu (Request):

+ HTTP (HyperText Transfer Protocol) – giao thức truyền tải siêu văn bản qua web.

+ RMI (Remote Method Invocation) – cho phép gọi phương thức từ xa giữa các ứng dụng Java.

- Nhiệm vụ chính: gửi yêu cầu từ client đến server.

2. Tầng Web (Web Tier-Tầng web):

- Chạy trên Web Container: môi trường của máy chủ ứng dụng (ví dụ Apache Tomcat, GlassFish Web Container) quản lý vòng đời các thành phần web.

- Các thành phần chính:

+ Servlet (Java Servlet) – thành phần Java xử lý yêu cầu HTTP và điều hướng luồng dữ liệu.

+ JSP (JavaServer Pages) – trang động trên server, nhúng Java vào HTML để sinh nội dung động.

+ JSF (JavaServer Faces) – framework dựa trên thành phần (component-based) để xây dựng giao diện web Java.

+ Filters (Servlet Filters) – bộ lọc xử lý trước/sau request, ví dụ: xác thực, log, nén dữ liệu.

- Nhiệm vụ chính:

+ Xử lý yêu cầu từ client.

+ Quản lý session (phiên làm việc).

+ Chuyển dữ liệu sang Business Tier (tầng nghiệp vụ

3. Tầng Business (Enterprise JavaBeans Tier-Tầng nghiệp vụ):

- Chạy trên EJB Container (Enterprise JavaBeans Container) – môi trường cung cấp dịch vụ quản lý vòng đời, giao dịch, bảo mật cho các thành phần nghiệp vụ.

- Các thành phần chính:

+ EJB (Enterprise JavaBeans) – thành phần Java chuẩn phục vụ logic nghiệp vụ, có thể phân tán, hỗ trợ giao dịch.

+ POJO (Plain Old Java Object) – đối tượng Java thuần túy, không phụ thuộc vào framework.

+ Spring Beans – thành phần quản lý bởi Spring Framework để triển khai nghiệp vụ thay cho EJB truyền thống.

- Nhiệm vụ chính:

+ Xử lý logic nghiệp vụ.

+ Giao tiếp với cơ sở dữ liệu (database).

+ Tích hợp với các dịch vụ khác hoặc hệ thống bên ngoài.

4. Tầng EIS (Enterprise Information System Tier-Tầng hệ thống thông tin doanh nghiệp):

- Thành phần chính:

+ Database (Cơ sở dữ liệu).

+ ERP (Enterprise Resource Planning) – hệ thống hoạch định tài nguyên doanh nghiệp.

+ CRM (Customer Relationship Management) – hệ thống quản lý quan hệ khách hàng.

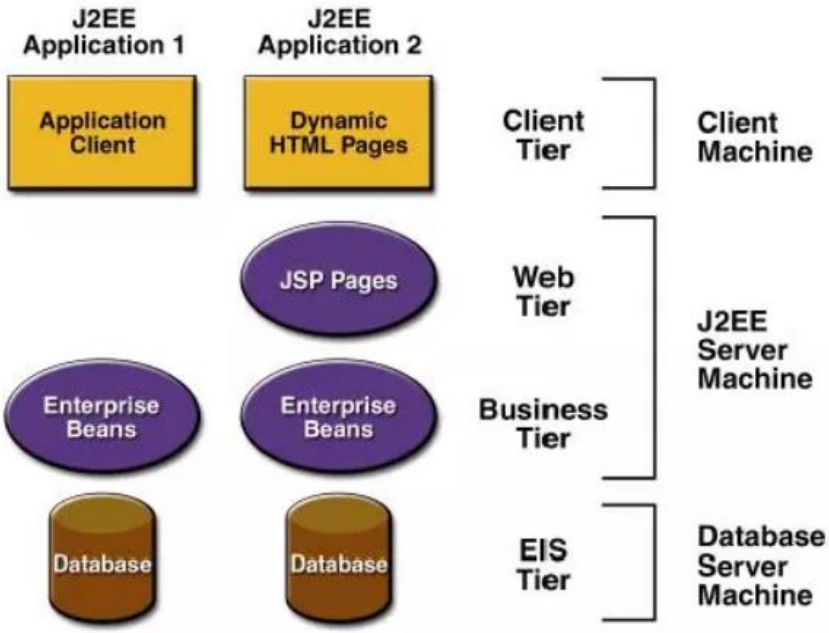
+ Dịch vụ web bên ngoài hoặc hệ thống tích hợp khác.

- Công nghệ kết nối:

+ JDBC (Java Database Connectivity) – chuẩn kết nối Java với cơ sở dữ liệu quan hệ.

+ JCA (Java Connector Architecture) – chuẩn kết nối Java với hệ thống thông tin doanh nghiệp (ERP, CRM, legacy systems).

+ JMS (Java Message Service) – chuẩn gửi/nhận thông điệp không đồng bộ giữa các thành phần ứng dụng.



Tóm lược luồng hoạt động

- Client (trình duyệt hoặc ứng dụng) gửi yêu cầu qua HTTP hoặc RMI.

- Web Tier nhận yêu cầu → Servlet/JSF/JSP xử lý sơ bộ.

- Business Tier thực hiện logic nghiệp vụ, giao tiếp cơ sở dữ liệu, hệ thống ngoài.

- EIS Tier cung cấp dữ liệu và dịch vụ nền tảng.

- Kết quả trả về cho Web Tier → hiển thị cho Client.

4.3.1. So sánh tổng quan Spring MVC và Struts

- **Spring** Framework: Spring là một framework toàn diện và phổ biến nhất hiện nay. Nó cung cấp một hệ sinh thái rộng lớn với nhiều module, bao gồm cả Spring MVC để xây dựng các ứng dụng web theo mô hình MVC, Spring Data để tương tác với cơ sở dữ liệu, và Spring Security để xử lý bảo mật. Sức mạnh cốt lõi của Spring giúp tạo ra các ứng dụng linh hoạt, dễ kiểm thử và bảo trì.

- **Struts**: Apache Struts là một trong những framework MVC đầu tiên và được sử dụng rộng rãi để phát triển các ứng dụng web J2EE. Struts giúp tách biệt logic nghiệp vụ (Model) khỏi giao diện (View) và luồng điều khiển (Controller), làm cho việc quản lý các ứng dụng lớn trở nên dễ

- So sánh tổng quan **Spring MVC** và **Struts**

Tiêu chí	Spring MVC	Struts (Struts 1/2)
Độ phổ biến	Phổ biến hơn hiện nay, liên tục được cập nhật	Xuất hiện sớm hơn, nhiều hệ thống cũ vẫn dùng
Kiến trúc	Dựa trên IoC (Inversion of Control), DI (Dependency Injection), hỗ trợ RESTful API	Mô hình MVC truyền thống, ít linh hoạt hơn REST
Cấu hình	Chủ yếu dùng annotation (@Controller, @RequestMapping…), giảm XML cấu hình	Cần nhiều file XML để cấu hình action/mapping
Tích hợp	Tích hợp tốt với các dự án Spring Boot, Spring Security, Hibernate	Ít tích hợp hơn, phải tự cấu hình nhiều
Hiệu năng	Tối ưu tốt hơn, nhẹ và nhanh	Chậm hơn một chút do kiến trúc cũ
Cộng đồng & hỗ trợ	Rộng, nhiều tutorial, nhiều plugin	Thu hẹp dần do xu hướng dịch chuyển sang Spring

4.3.2. Các thành phần cơ bản trong Spring MVC

Spring MVC hoạt động dựa trên một vài thành phần cốt lõi để xử lý các yêu cầu HTTP.

- **DispatcherServlet:** Là Controller trung tâm trong Spring MVC. Nó nhận tất cả các yêu cầu từ người dùng, điều phối chúng tới các Controller thích hợp, và cuối cùng trả về kết quả.

- **Controller:** Là các lớp Java được đánh dấu bằng `@Controller`. Chúng chứa các phương thức xử lý yêu cầu (`@RequestMapping`) và trả về `ModelAndView` hoặc một tên View.

- **Model:** Chứa dữ liệu mà View cần hiển thị.

- **View:** Thường là các tệp JSP, Thymeleaf, hoặc HTML, có nhiệm vụ hiển thị dữ liệu từ Model.

- **HandlerMapping:** Liên kết một yêu cầu đến phương thức Controller phù hợp dựa trên URL, phương thức HTTP, v.v.

- **ViewResolver:** Giúp DispatcherServlet tìm đúng View để hiển thị.

4.3.3. Các thành phần cơ bản trong Struts

Struts có một kiến trúc phân cấp để xử lý các yêu cầu.

- **ActionServlet (Struts 1) hoặc FilterDispatcher (Struts 2):** Là Controller trung tâm. Nó nhận yêu cầu, tìm `Action` phù hợp, và chuyển giao việc xử lý.

- **Action:** Là các lớp Java xử lý logic nghiệp vụ. Khi một yêu cầu được gửi đến, Struts sẽ gọi phương thức `execute()` trong lớp Action tương ứng.

- **ActionForm (Struts 1) hoặc Action (Struts 2):** Dùng để lưu trữ dữ liệu từ Form của người dùng.

- **Result:** Định nghĩa nơi chuyển hướng sau khi Action được thực thi (ví dụ: một trang JSP, một Action khác, hoặc một URL).

- **struts-config.xml:** File XML cấu hình chính của Struts, nơi bạn định nghĩa các Action, Form, và các chuyển hướng (`<forward>`).

4.3.4. Khi nào chọn Spring, khi nào chọn Struts

Việc lựa chọn giữa Spring MVC và Struts phụ thuộc vào yêu cầu của dự án và kinh nghiệm của đội ngũ phát triển.

- Nên chọn Spring MVC khi:

+ Bạn cần một framework **linh hoạt và hiện đại** với cộng đồng lớn và cập nhật thường xuyên.

+ Bạn muốn tận dụng các tính năng mạnh mẽ khác của hệ sinh thái Spring như **Dependency Injection**, **Spring Security**, **Spring Data**.

+ Dự án yêu cầu phát triển nhanh, đặc biệt với sự hỗ trợ của **Spring Boot**.

+ Bạn muốn sử dụng cấu hình dựa trên chú thích (Annotation) thay vì XML.

- Nên chọn Struts khi:

+ Bạn đang làm việc trên một **dự án kế thừa** (legacy project) đã sử dụng Struts và cần duy trì.

+ Bạn muốn một framework có cấu trúc và quy trình được định nghĩa sẵn một cách chặt chẽ.

+ Đội ngũ phát triển đã có kinh nghiệm sâu rộng với Struts và muốn tiếp tục sử dụng nó.

Ngày nay, **Spring MVC** (đặc biệt là với Spring Boot) đã trở thành lựa chọn hàng đầu cho các dự án mới nhờ vào tính linh hoạt, khả năng mở rộng, và sự hỗ trợ mạnh mẽ của cộng đồng.

4.4. Nền tảng Spring MVC (hoặc Struts)

4.4.1. Kiến trúc và nguyên lý hoạt động

Spring MVC là một framework dựa trên mô hình MVC, cung cấp một cách có cấu trúc để xây dựng ứng dụng web.

Nguyên lý hoạt động chính:

- **Request:** Người dùng gửi một yêu cầu (request) đến máy chủ.

- **DispatcherServlet:** Yêu cầu này được nhận bởi **DispatcherServlet** - "trái tim" của Spring MVC. Nó hoạt động như một bộ điều khiển trung tâm, điều phối toàn bộ luồng xử lý.

- **Controller:** DispatcherServlet tìm kiếm một Controller phù hợp với yêu cầu.

- **Business Logic:** Controller xử lý yêu cầu, tương tác với các lớp nghiệp vụ (service, repository) để lấy và xử lý dữ liệu.

- **Model:** Dữ liệu sau khi xử lý được đóng gói vào một đối tượng **Model**.

- **View:** Controller chọn một **View** phù hợp để hiển thị dữ liệu.

- **Response:** Cuối cùng, View trả lại một trang HTML (hoặc định dạng khác) về cho người dùng.

4.4.2. Cấu hình dự án Spring MVC / Struts

❖ **Spring MVC**

- **web.xml**: khai báo DispatcherServlet.

- **spring-servlet.xml**: khai báo bean Controller, ViewResolver…

- Hoặc dùng **Java-based config** với `@Configuration` + `@EnableWebMvc`.

- Khai báo thư viện Spring (Maven/Gradle).

Ví dụ web.xml:

```
<servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

❖ **Struts**

- **web.xml**: khai báo Struts Filter.

- **struts.xml**: định nghĩa Action, Result, Interceptor.

- Khai báo thư viện Struts (Maven/Gradle).

Ví dụ web.xml:

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-
class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

4.4.3. Controller, Model, View trong Spring MVC / Struts

- Controller: Là các lớp Java được đánh dấu bằng @Controller. Chúng có nhiệm vụ xử lý các yêu cầu từ người dùng.

Mỗi phương thức trong Controller được đánh dấu bằng @RequestMapping để liên kết với một URL cụ thể.

- Model: Là một đối tượng trong Spring MVC dùng để lưu trữ dữ liệu. Dữ liệu này được truyền từ Controller sang View để hiển thị.

Bạn có thể thêm dữ liệu vào Model bằng cách sử dụng đối tượng Model trong phương thức của Controller.

- View: Là giao diện người dùng, thường là các file JSP, HTML. Nó nhận dữ liệu từ Model và hiển thị lên trình duyệt.

Spring MVC sử dụng View Resolver để tìm đúng View từ tên được trả về bởi Controller.

❖ **Spring MVC**

java

```
@Controller
public class SinhVienController {
    @RequestMapping("/list")
    public String list(Model model) {
        model.addAttribute("ds", sinhVienService.getAll());
        return "listSinhVien"; // JSP
    }
}
```

❖ **Struts**

xml

```
<action name="listSV" class="com.example.SinhVienAction" method="list">
  <result name="success">/listSinhVien.jsp</result>
</action>
```

4.4.4. Mapping URL, xử lý request và response

- Mapping URL: Bạn sử dụng chú thích @RequestMapping hoặc các biến thể cụ thể hơn như @GetMapping, @PostMapping để liên kết một phương thức Controller với một URL.

- Xử lý Request: Dữ liệu từ request có thể được lấy bằng nhiều cách:

`@RequestParam`: Lấy giá trị từ tham số URL.

`@PathVariable`: Lấy giá trị từ biến trong URL.

`@RequestBody`: Chuyển đổi dữ liệu JSON từ request thành đối tượng Java.

- Xử lý Response:

Trả về chuỗi: Controller trả về tên của View (ví dụ: "students"), sau đó View Resolver sẽ tìm file students.jsp.

Trả về JSON/XML: Sử dụng @ResponseBody hoặc @RestController để trả về dữ liệu trực tiếp, phù hợp cho các ứng dụng API.

Spring MVC ví dụ:

Java:

```
@PostMapping("/add")
public String add(@ModelAttribute SinhVien sv) {
    sinhVienService.add(sv);
    return "redirect:/list";
```

}

Struts ví dụ:

Xml:

```
<action name="addSV" class="com.example.SinhVienAction" method="add">
  <result name="success">/listSinhVien.jsp</result>
</action>
```

4.4.5. Quản lý form, validation dữ liệu người dùng

- Form Handling: Spring MVC tự động liên kết dữ liệu từ form HTML với một đối tượng Java (gọi là Form Backing Object).

- Validation: Spring hỗ trợ validation bằng cách sử dụng chú thích của Bean Validation API (như @NotNull, @Size, @Email).

+ Thêm chú thích validation vào đối tượng Form.

+ Sử dụng `@Valid` trong phương thức Controller để kích hoạt quá trình validation.

+ Sử dụng đối tượng `BindingResult` để kiểm tra kết quả validation và xử lý các lỗi.

4.4.6. Thực hành: xây dựng một ứng dụng web CRUD đơn giản

Yêu cầu: Quản lý danh sách sinh viên: thêm, sửa, xóa, xem.

Các bước với Spring MVC

1.Tạo dự án Maven Spring MVC.

2.Cấu hình DispatcherServlet + ViewResolver.

3.Tạo **Model**: SinhVien.java.

4.Tạo **DAO/Service**: CRUD với DB hoặc file XML.

5.Tạo **Controller**: mapping các URL /list, /add, /edit, /delete.

6.Tạo **View JSP**: form nhập, list hiển thị.

7.Thêm **validation** khi nhập dữ liệu.

Các bước với Struts

1.Tạo dự án Maven Struts2.

2.Cấu hình StrutsPrepareAndExecuteFilter trong web.xml.

3.Khai báo action trong struts.xml.

4.Tạo lớp **Action** để xử lý CRUD.

5.Tạo JSP dùng Struts Taglibs để hiển thị và nhập dữ liệu.

Cụ thể:

- Model: Tạo lớp Student.java với các thuộc tính (id, name, email).

- Controller: Tạo StudentController.java với các phương thức:

`@GetMapping("/students")`: Hiển thị danh sách sinh viên.

`@GetMapping("/students/add")`: Hiển thị form thêm sinh viên.

`@PostMapping("/students/add")`: Xử lý việc thêm sinh viên.

`@GetMapping("/students/edit/{id}")`: Hiển thị form chỉnh sửa.

`@PostMapping("/students/update")`: Xử lý việc cập nhật.

`@GetMapping("/students/delete/{id}")`: Xử lý việc xóa sinh viên.

- View: Tạo các file JSP tương ứng (list-students.jsp, student-form.jsp) để hiển thị dữ liệu và form.

- Service/Repository: Tạo các lớp để xử lý logic nghiệp vụ và tương tác với cơ sở dữ liệu.

4.5. Tích hợp với các thành phần khác

4.5.1. Kết nối cơ sở dữ liệu với JDBC / JPA / Hibernate

So sánh các công nghệ kết nối cơ sở dữ liệu

Công nghệ	Viết tắt	Ý nghĩa đầy đủ	Chức năng chính	Khi nên sử dụng
JDBC	**J**ava **D**ata**b**ase **C**onnectivity	Chuẩn giao tiếp với cơ sở dữ liệu trong Java. Là API gốc giúp ứng dụng Java kết nối, gửi truy vấn SQL và nhận kết quả từ DB.	Kết nối trực tiếp tới DB, thực thi lệnh SQL (SELECT, INSERT…), xử lý ResultSet, quản lý transaction thủ công.	Ứng dụng nhỏ, cần kiểm soát chi tiết SQL, ít lớp trung gian.
JPA	**J**ava **P**ersistence **A**PI	Chuẩn giao tiếp ORM trong Java. Cho phép lập trình viên thao tác DB qua đối tượng Java thay vì SQL thuần.	Định nghĩa Entity, mapping giữa class và bảng dữ liệu, hỗ trợ CRUD, transaction tự động.	Ứng dụng trung bình hoặc lớn, muốn giảm viết SQL thủ công, chuẩn hóa ORM.
Hibernate	– (Tên riêng, nhưng triển khai JPA)	Framework ORM mạnh mẽ, triển khai đầy đủ JPA, đồng thời mở rộng nhiều tính năng nâng cao.	Tự động ánh xạ đối tượng – bảng, HQL (Hibernate Query Language), cache, lazy loading, transaction management nâng cao.	Ứng dụng cần tính năng ORM nâng cao hơn JPA chuẩn, muốn nhiều công cụ hỗ trợ sẵn.

ORM (Object–Relational Mapping): Kỹ thuật ánh xạ giữa class Java và bảng trong DB để thao tác dữ liệu bằng đối tượng thay vì SQL.

API (Application Programming Interface): Giao diện lập trình ứng dụng, tập hợp các hàm hoặc lớp cho phép lập trình viên tương tác với hệ thống khác.

Entity: Lớp Java đại diện cho một bảng dữ liệu.

Ví dụ: kết nối dữ liệu với JDBC:

```
Connection conn = DriverManager.getConnection(url, user, pass);
PreparedStatement ps = conn.prepareStatement("SELECT * FROM sinhvien");
ResultSet rs = ps.executeQuery();
```

Ví dụ: kết nối dữ liệu với Hibernate:

```
Session session = sessionFactory.openSession();
List<SinhVien> ds = session.createQuery("from SinhVien").list();
```

4.5.2. Sử dụng XML hoặc Annotation để cấu hình bean/service

Kiểu cấu hình	Đặc điểm
XML-based configuration	Cấu hình trong file `applicationContext.xml`. Khai báo bean, datasource, transaction manager…
Annotation-based configuration	Dùng `@Component`, `@Service`, `@Repository`, `@Autowired` trong code Java. Linh hoạt hơn, ít XML.

Ví dụ XML:

```
<bean id="sinhVienService" class="com.example.SinhVienServiceImpl">
  <property name="sinhVienDAO" ref="sinhVienDAO"/>
```

```
</bean>
```

Ví dụ Annotation:

```
@Service
public class SinhVienServiceImpl implements SinhVienService {
    @Autowired
    private SinhVienDAO sinhVienDAO;
}
```

4.5.3. Tích hợp bảo mật cơ bản (Spring Security / Filter)

❖ Với Spring Security

- Là module bảo mật tích hợp sẵn trong Spring.
- Hỗ trợ Authentication (xác thực), Authorization (phân quyền), CSRF protection.
- Cấu hình qua `SecurityConfig.java` hoặc `security.xml`.
- Có thể dùng JDBC/LDAP để lưu user, role.

Ví dụ config đơn giản bằng Java:

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
          .authorizeRequests()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/**").permitAll()
            .and()
          .formLogin();
    }
}
```

❖ Với Filter tự viết

- Dùng **Servlet Filter** để kiểm tra session/login trước khi cho vào Controller/JSP.
- Filter chạy trước request để chặn hoặc cho phép.

Ví dụ Filter:

```
@WebFilter("/admin/*")
public class AuthFilter implements Filter {
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) req;
        HttpSession session = request.getSession(false);
        if (session != null && session.getAttribute("user") != null) {
            chain.doFilter(req, res);
        } else {
            request.getRequestDispatcher("/login.jsp").forward(req, res);
        }
    }
}
```

4.6. Thực hành ứng dụng web hoàn chỉnh

❖ Xây dựng ứng dụng quản lý sinh viên

- Chọn Spring MVC hoặc Struts làm nền tảng.
- Tạo project trong IDE (IntelliJ/NetBeans/Eclipse).
- Tạo các lớp Model (SinhVien), Controller, View (JSP).
- Cấu hình cơ sở dữ liệu (MySQL/PostgreSQL).

❖ **Thực hiện CRUD + xác thực/ phân quyền**

- CRUD: Thêm – Xem – Sửa – Xóa sinh viên.
- Tích hợp Spring Security (Spring) hoặc Filter/Interceptor (Struts) để:

+ Đăng nhập (Login).
+ Xác thực quyền người dùng (Admin/User).
+ Phân quyền chức năng.

❖ **Báo cáo / Export dữ liệu**

- Sử dụng thư viện iText / JasperReports để xuất PDF.
- Sử dụng thư viện Apache POI để xuất Excel.
- Tạo nút "Export" trên giao diện để tải file.

❖ **Triển khai ứng dụng lên Server**

- Cấu hình file WAR (Web Application Archive).
- Deploy lên Apache Tomcat hoặc GlassFish.
- Kiểm tra ứng dụng trên trình duyệt.

4.7. Tổng kết và mở rộng

❖ **Đánh giá ưu – nhược điểm của từng nền tảng**

Nền tảng	Ưu điểm	Nhược điểm
Spring MVC	- Kiến trúc hiện đại, tách biệt rõ Model–View–Controller. - Hệ sinh thái mạnh (Spring Boot, Spring Security, Spring Data). - Tích hợp dễ dàng với nhiều công nghệ khác (Hibernate, JPA).	- Hơi phức tạp khi mới học. - Cần hiểu rõ cấu hình XML hoặc Annotation.
Struts	- Cũ nhưng ổn định, đã có nhiều dự án thực tế. - Dễ tiếp cận hơn với người mới. - Nhiều tài liệu cũ phong phú.	- Công nghệ cũ, ít được cập nhật. - Không linh hoạt bằng Spring. - Bảo mật không tích hợp sẵn như Spring Security.

❖ **Xu hướng phát triển framework web Java hiện nay**

1. Spring Boot: Cấu hình tối giản và nhanh chóng

Spring Boot là một trong những xu hướng quan trọng nhất hiện nay. Nó ra đời để giải quyết những khó khăn trong việc cấu hình dự án Spring truyền thống, giúp việc phát triển ứng dụng trở nên nhanh chóng và đơn giản hơn rất nhiều.

- **Cấu hình tối giản (Convention over Configuration):** Spring Boot tự động cấu hình gần như mọi thứ dựa trên các thư viện (dependencies) bạn thêm vào. Điều này giúp giảm đáng kể lượng code cấu hình XML hoặc Java.

- **Tích hợp sẵn Server:** Spring Boot đi kèm với một máy chủ nhúng (embedded server) như **Tomcat**, Jetty, hoặc Undertow. Bạn có thể chạy ứng dụng chỉ bằng một câu lệnh `java -jar`, không cần phải cài đặt và cấu hình máy chủ bên ngoài.

- **Hướng đi mới cho Spring MVC:** Spring Boot là cách tiếp cận hiện đại để xây dựng ứng dụng với Spring MVC. Nó tích hợp chặt chẽ Spring MVC, giúp bạn có thể tạo một ứng dụng web hoàn chỉnh chỉ trong vài phút.

2. Jakarta EE: Phiên bản tiếp theo của J2EE

Jakarta EE là tên mới của **Java EE** sau khi Oracle chuyển giao cho Tổ chức Eclipse. Jakarta EE tiếp tục phát triển các API chuẩn cho doanh nghiệp, cung cấp các nền tảng mạnh mẽ và được hỗ trợ bởi nhiều nhà cung cấp.

- **Nâng cấp và cải tiến:** Jakarta EE tập trung vào việc hiện đại hóa các API truyền thống, tối ưu hóa cho môi trường cloud và kiến trúc Microservices.

- **Sự cạnh tranh và bổ sung:** Mặc dù Spring Boot rất phổ biến, Jakarta EE vẫn là một lựa chọn đáng tin cậy, đặc biệt trong môi trường doanh nghiệp truyền thống, nơi yêu cầu các tiêu chuẩn và khả năng tương thích cao.

3. Microservices: Chia nhỏ ứng dụng thành các dịch vụ

Microservices là một kiến trúc phần mềm, trong đó một ứng dụng lớn được chia thành các dịch vụ nhỏ, độc lập và có thể triển khai riêng biệt. Xu hướng này ngày càng phổ biến và Java có nhiều công cụ mạnh mẽ để hỗ trợ.

- **Spring Cloud:** Là một bộ công cụ mạnh mẽ trong hệ sinh thái Spring để xây dựng và quản lý các kiến trúc Microservices. Nó cung cấp các tính năng như Service Discovery (Eureka), Circuit Breaker (Hystrix), và API Gateway.

- **Lợi ích:** Kiến trúc này giúp ứng dụng dễ bảo trì, mở rộng và phát triển độc lập từng phần.

4. Reactive Programming: Lập trình phản ứng và phi đồng bộ

Reactive Programming là một mô hình lập trình tập trung vào luồng dữ liệu và sự thay đổi. Nó rất phù hợp cho các ứng dụng cần xử lý lượng lớn dữ liệu hoặc có độ trễ cao.

Spring WebFlux: Là một phần của Spring Framework, hỗ trợ lập trình phản ứng và phi đồng bộ. WebFlux sử dụng các luồng phản ứng (reactive streams) để xử lý các yêu cầu một cách hiệu quả hơn, giúp tối ưu hóa việc sử dụng tài nguyên của máy chủ.

5. Kết hợp với Frontend hiện đại: Xây dựng API

Các ứng dụng web Java hiện đại không chỉ tạo ra các trang HTML mà còn xây dựng các API (Application Programming Interface). Các API này được sử dụng để cung cấp dữ liệu cho các framework **Frontend** mạnh mẽ như: **Angular, React,Vue.js**

Cách tiếp cận này giúp tách biệt hoàn toàn phần giao diện người dùng (Frontend) khỏi phần xử lý logic (Backend), cho phép các nhóm phát triển làm việc độc lập và hiệu quả hơn. Spring Boot là lựa chọn hàng đầu để xây dựng các **API RESTful** cho mục đích này.

TÀI LIỆU THAM KHẢO

❖ **HTML & CSS & JavaScript**

- W3Schools: Trang "kinh điển", dễ học, có ví dụ chạy trực tiếp, cực hợp cho người mới.
- CSS-Tricks: Đặc biệt hữu ích cho CSS nâng cao và các "tips" về responsive web.

❖ **Java:**

- Oracle Java Tutorials**:** Tài liệu chính thức của Oracle - chuẩn mực và cập nhật nhất.
- Java highlight: Tài liệu Tiếng Việt hướng dẫn đầy đủ về lập trình Java
- W3Schools:Trang "kinh điển", dễ học, có ví dụ chạy trực tiếp, cực hợp cho người mới.
- GeeksforGeeks – Java**:** Giải thích chi tiết kèm ví dụ, từ cơ bản tới nâng cao.

HƯỚNG DẪN CÀI ĐẶT CÔNG CỤ THỰC HÀNH LẬP TRÌNH WEB

1. Giới thiệu công cụ/phần mềm phục vụ lập trình web

- **JDK** *(phiên bản 17):* (Java Development Kit) là bộ công cụ phát triển phần mềm bằng ngôn ngữ Java

- **NetBeans** *(Phiên bản 21):* là một IDE (Integrated Development Environment), môi trường phát triển tích hợp – miễn phí và mã nguồn mở, dùng để phát triển ứng dụng Java (và cả các ngôn ngữ khác như C/C++, PHP, HTML/JS, v.v).

- **Apache TomCat** *(Phiên bản 11):* Máy chủ web mã nguồn mở giúp chạy ứng dụng web bằng Java (Servlet, JSP).

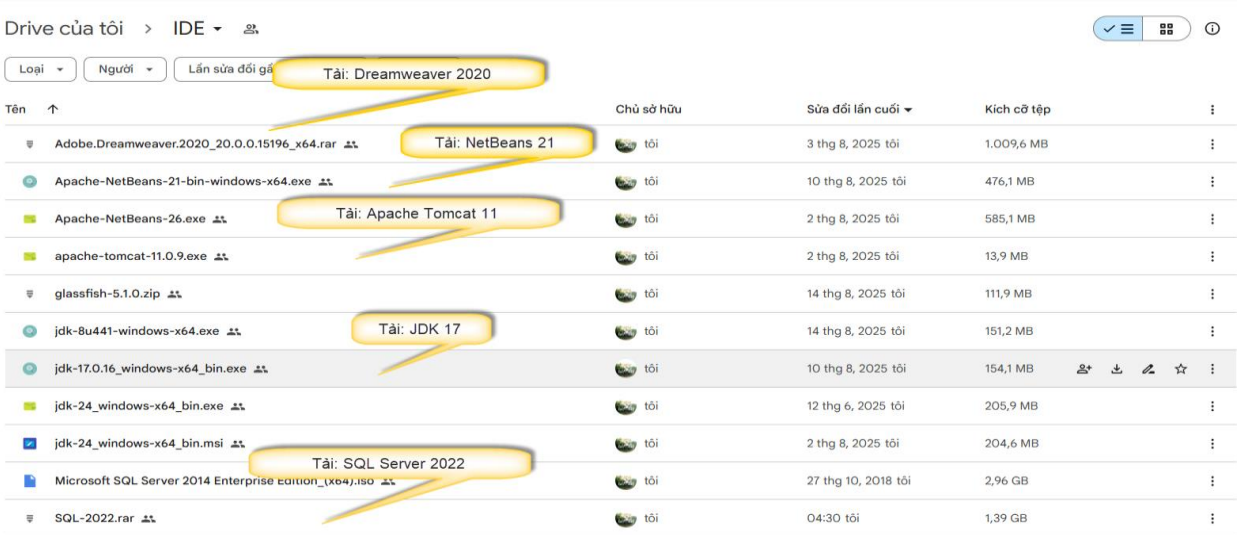
- **Dreamweaver**: một phần mềm thiết kế và phát triển web do Adobe phát triển. Nó hỗ trợ lập trình viên và nhà thiết kế tạo các website tĩnh và động với giao diện kéo thả (Design view) kết hợp với soạn mã nguồn (Code view).

- **SQL Server**: Hệ quản trị cơ sở dữ liệu

2. Tải phần mềm

- Để tránh có sự tương thích giữa các công cụ (công nghệ) phần mềm, tránh tình trạng xảy ra lỗi khi cài đặt và chạy ứng dụng, cần tải các phiên bản gợi ý nêu trên.

- Tải các công cụ phần mềm tại địa chỉ: Link

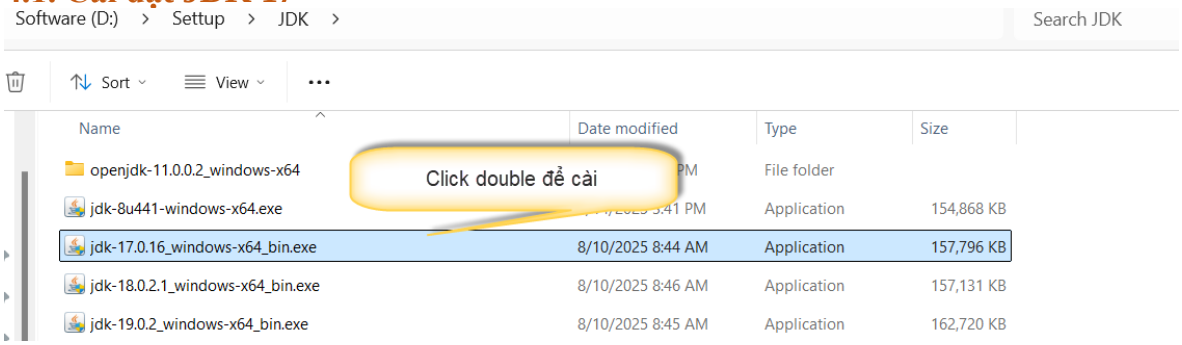


3. Trình tự cài đặt phần mềm

- Bước 1: Cài đặt **JDK 17**
- Bước 2: Cài đặt **NetBeans 21**
- Bước 3: Cài đặt **Apache TomCat 11**
- Bước 4: Tích hợp **Apache TomCat** trong **NetBeans**
- Bước 5: Cài đặt Dreamweaver (Tham cài đặt thêm để hỗ trợ lập trình và thiết kế giao diện)
- Bước 6: Cài đặt SQL Server

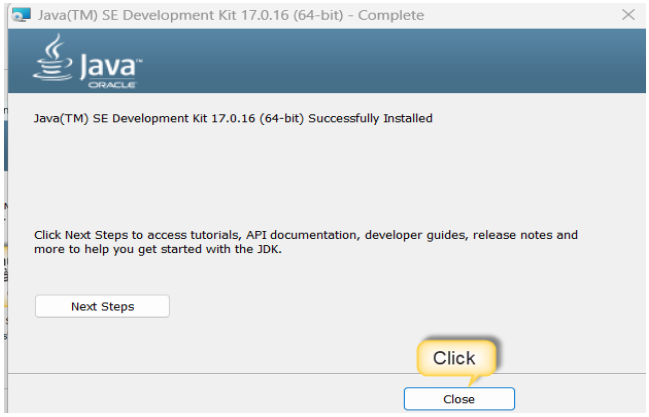
4. Quá trình cài đặt phần mềm

4.1. Cài đặt JDK 17

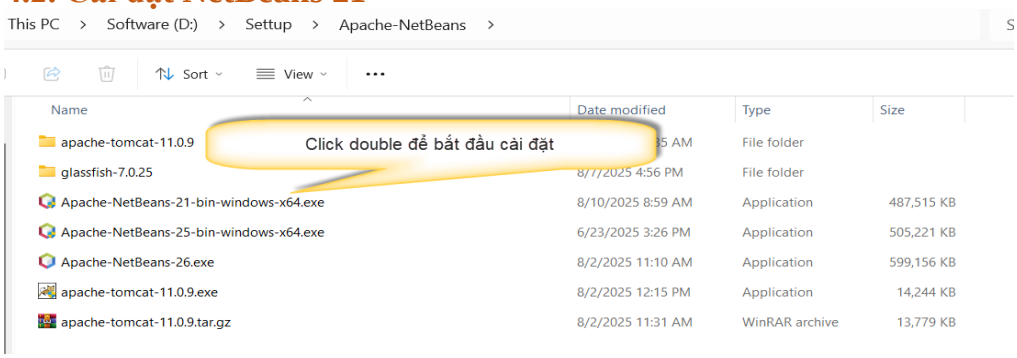


- Chọn **Yes**

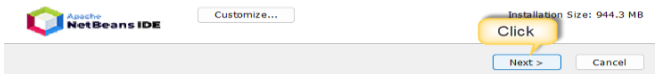
Quá trình cài đặt sẽ hoàn tất trong ít phút

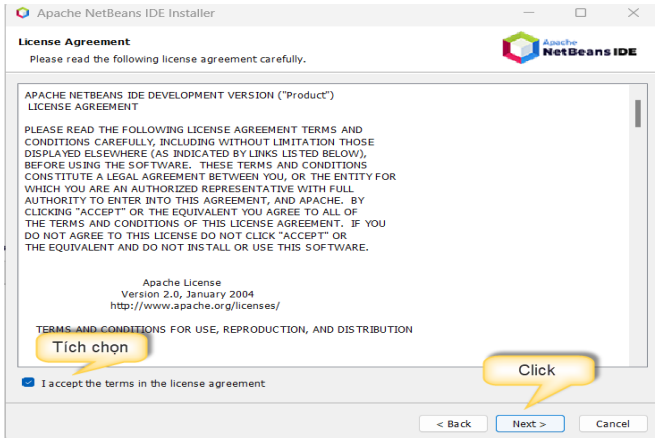


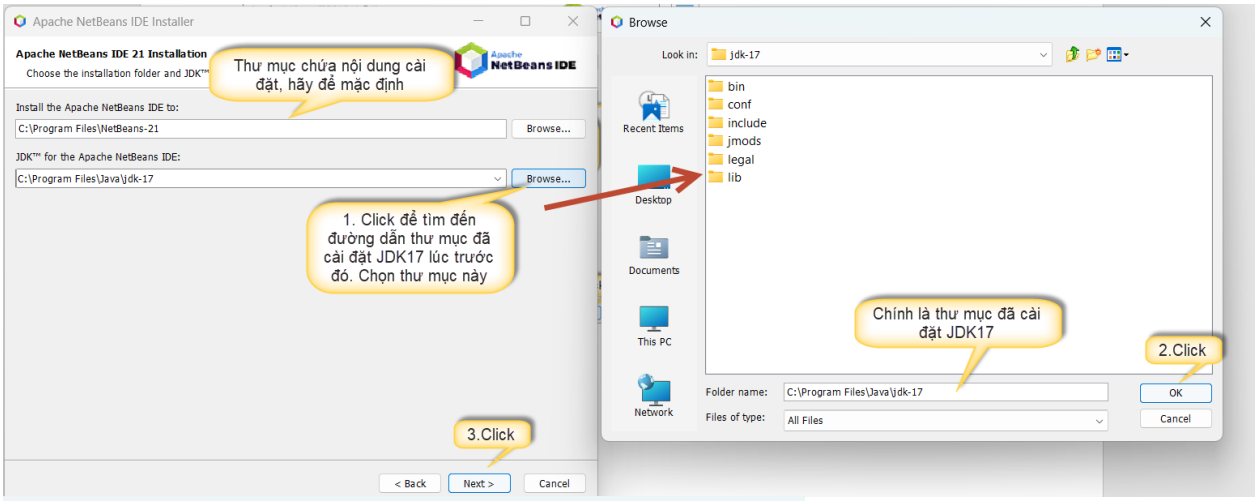
4.2. Cài đặt NetBeans 21



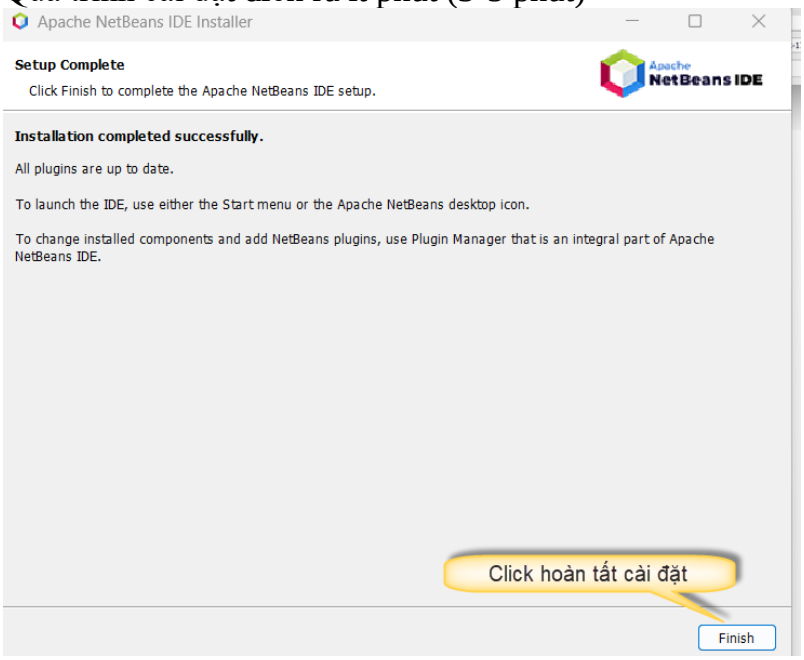
- Chọn **Yes**
- Chờ cho đến khi xuất hiện cửa sổ dưới đây



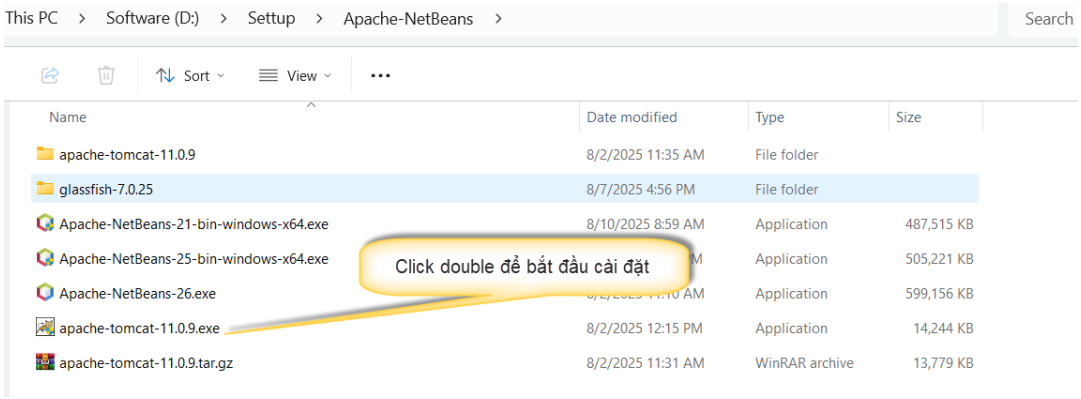




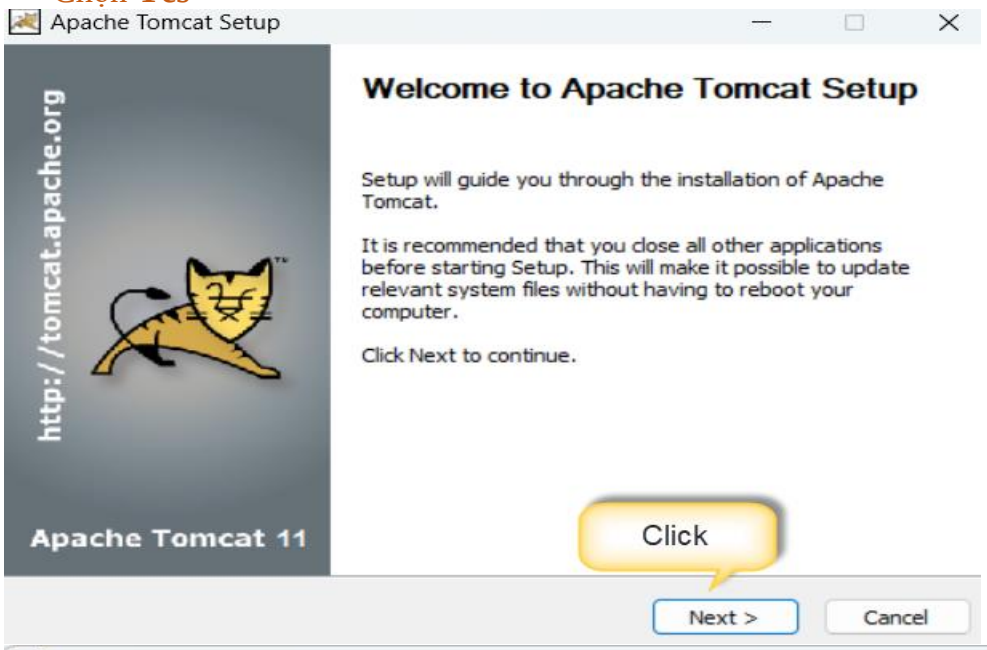
Quá trình cài đặt diễn ra ít phút (3-5 phút)

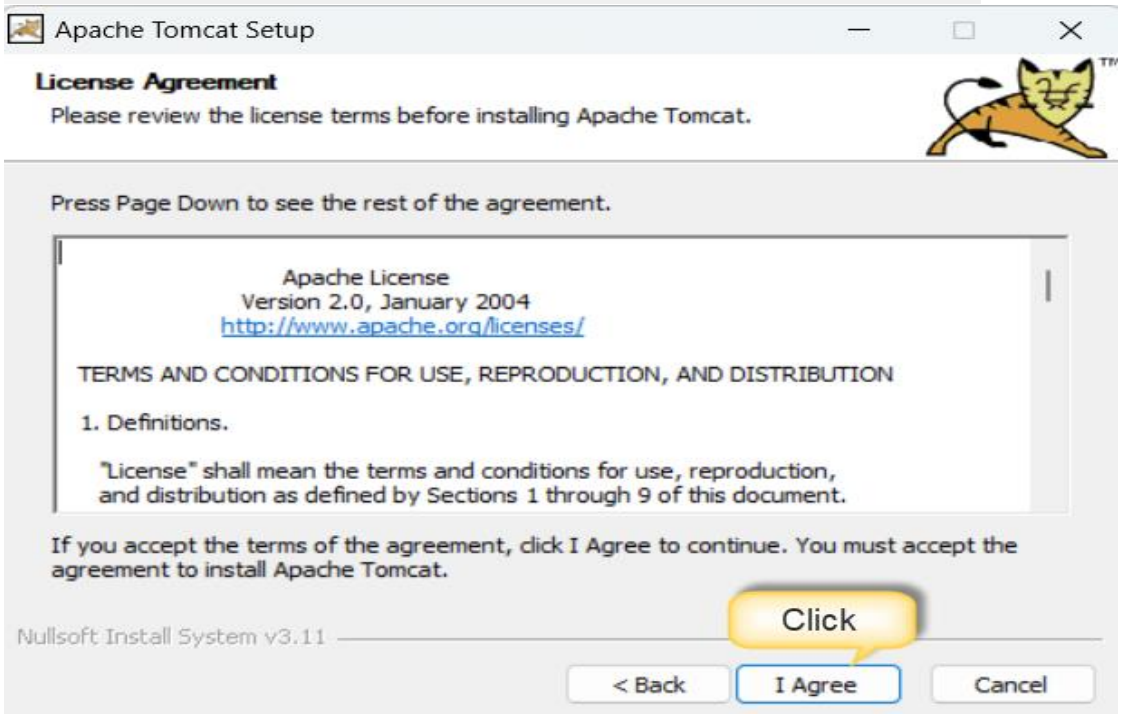


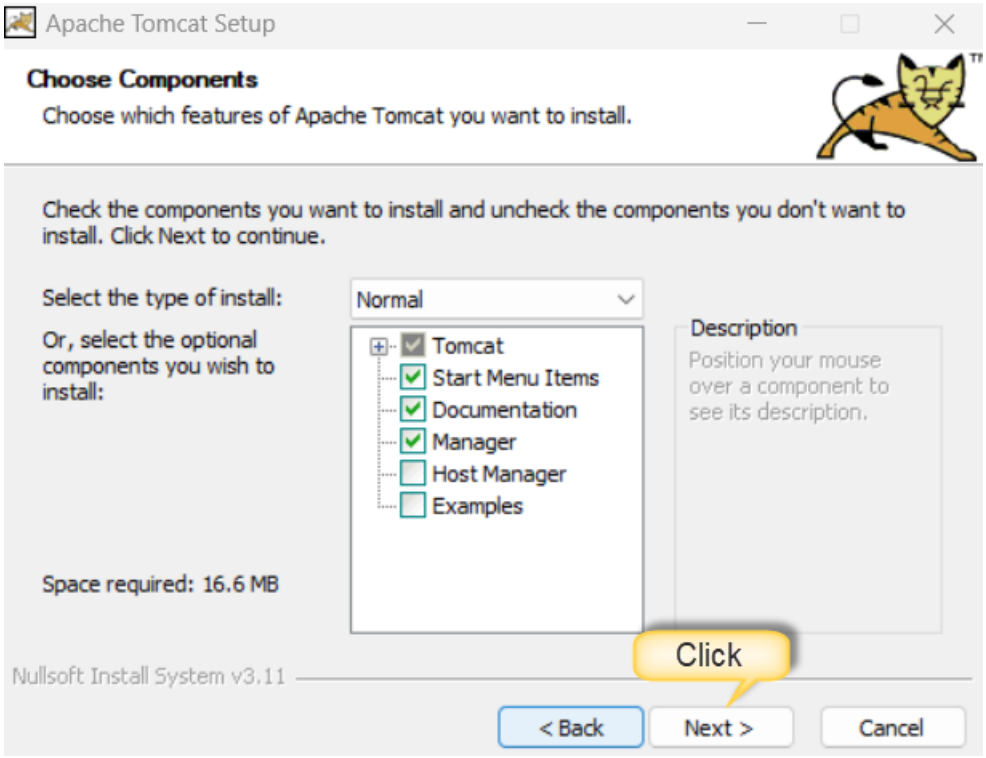
4.3. Cài đặt Apache Tomcat 11

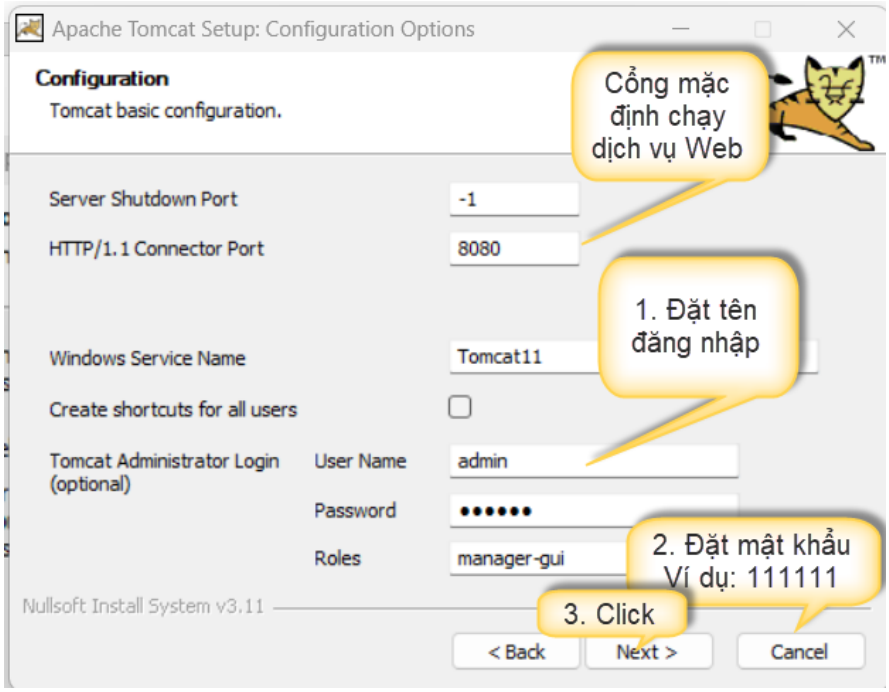

This PC › Software (D:) › Settup › Apache-NetBeans ›
Search
Sort
View
Name
Date modified
Type
Size
apache-tomcat-11.0.9
8/2/2025 11:35 AM
File folder
glassfish-7.0.25
8/7/2025 4:56 PM
File folder
Apache-NetBeans-21-bin-windows-x64.exe
8/10/2025 8:59 AM
Application
487,515 KB
Apache-NetBeans-25-bin-windows-x64.exe
Application
505,221 KB
Apache-NetBeans-26.exe
Application
599,156 KB
apache-tomcat-11.0.9.exe
8/2/2025 12:15 PM
Application
14,244 KB
apache-tomcat-11.0.9.tar.gz
8/2/2025 11:31 AM
WinRAR archive
13,779 KB
Click double để bắt đầu cài đặt

- Chọn **Yes**

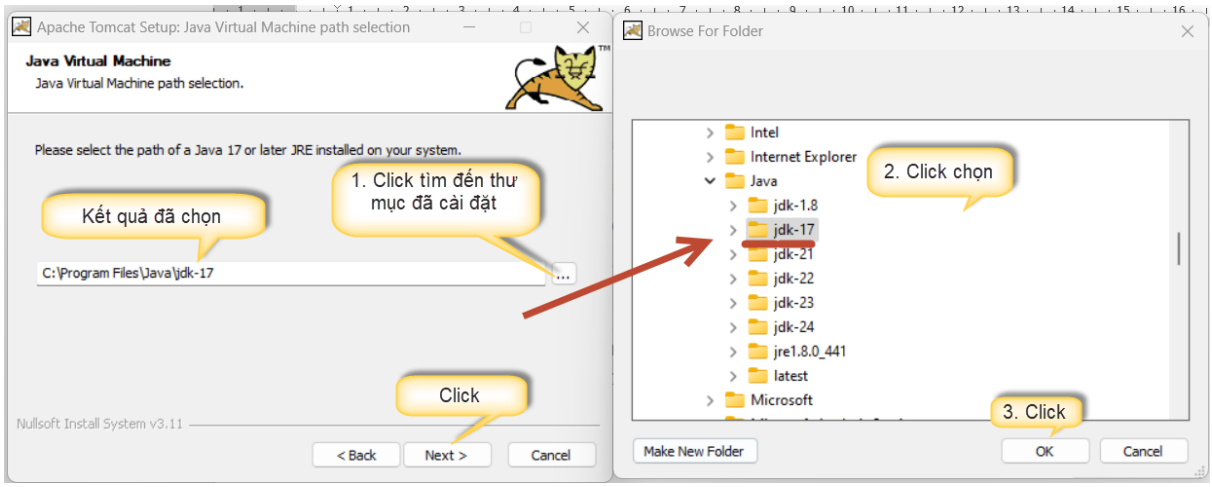

Apache Tomcat Setup
http://tomcat.apache.org
Apache Tomcat 11
Welcome to Apache Tomcat Setup
Setup will guide you through the installation of Apache Tomcat.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Click
Next >
Cancel

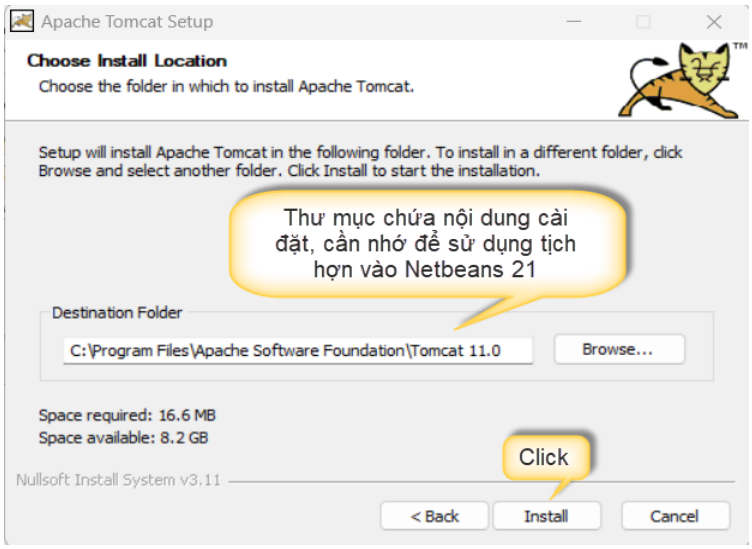

Apache Tomcat Setup
License Agreement
Please review the license terms before installing Apache Tomcat.
Press Page Down to see the rest of the agreement.
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Apache Tomcat.
Nullsoft Install System v3.11
Click
< Back
I Agree
Cancel



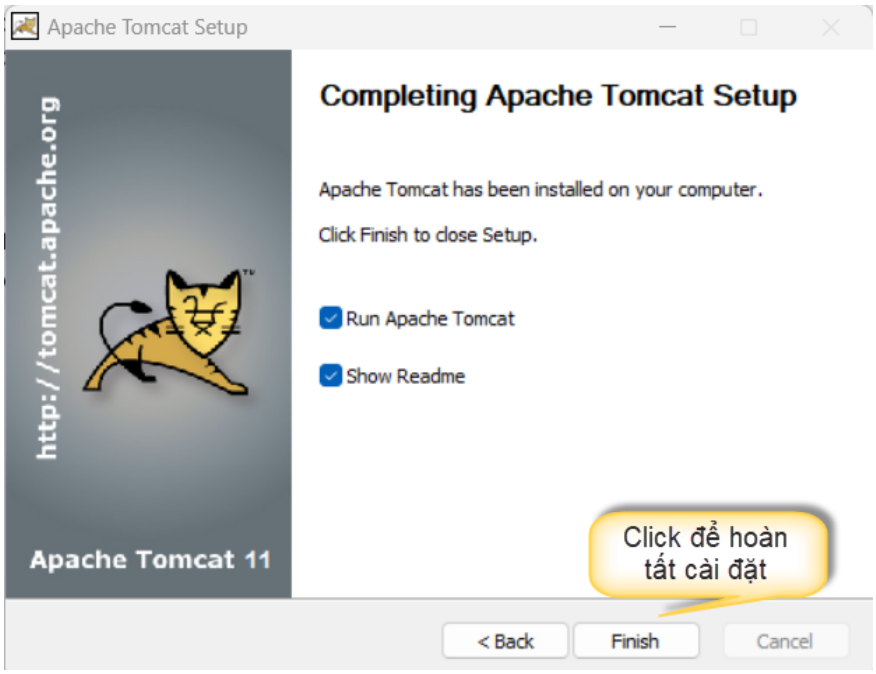


Hãy đặt tên đăng nhập và mật khẩu để quản trị TomCat, sử dụng để chạy dịch vụ web từ NetBeans. Cần nhớ tên đăng nhập và mật khẩu để sử dụng về sau.

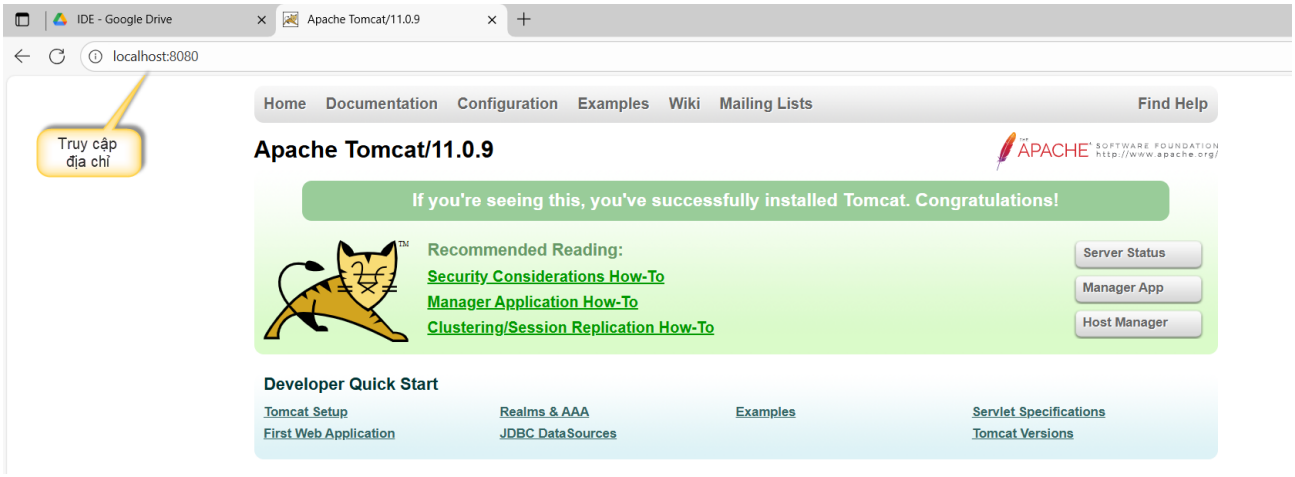




- Cần nhớ thư cài đặt Tomcat 11 để sử dụng cho việc tích hợp vào NetBeans 11 về sau
- Chờ ít phút chương trình cài đặt hoàn tất (1-2 Phút)

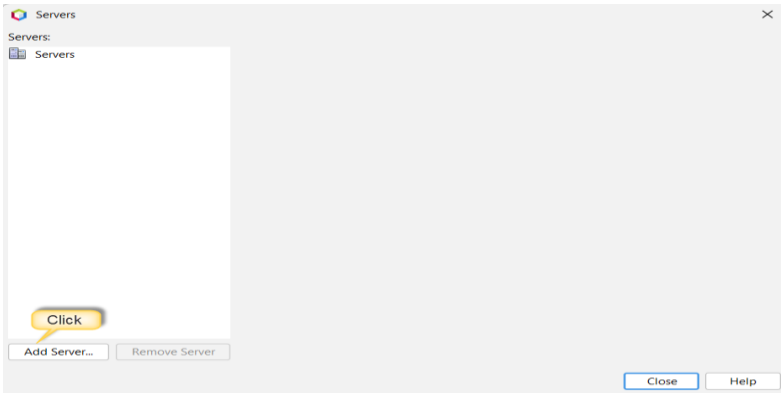


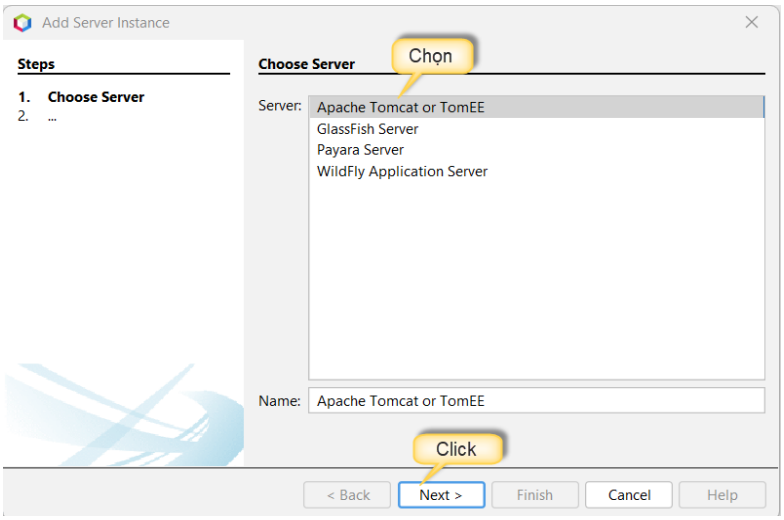
- Sau khi hoàn tất, mở trình duyệt web, gõ địa chỉ http://localhost:8080 để kiểm tra kết quả cài đặt Tomcat 11. Nếu hiện thị như dưới đây thì đã cài đặt thành công

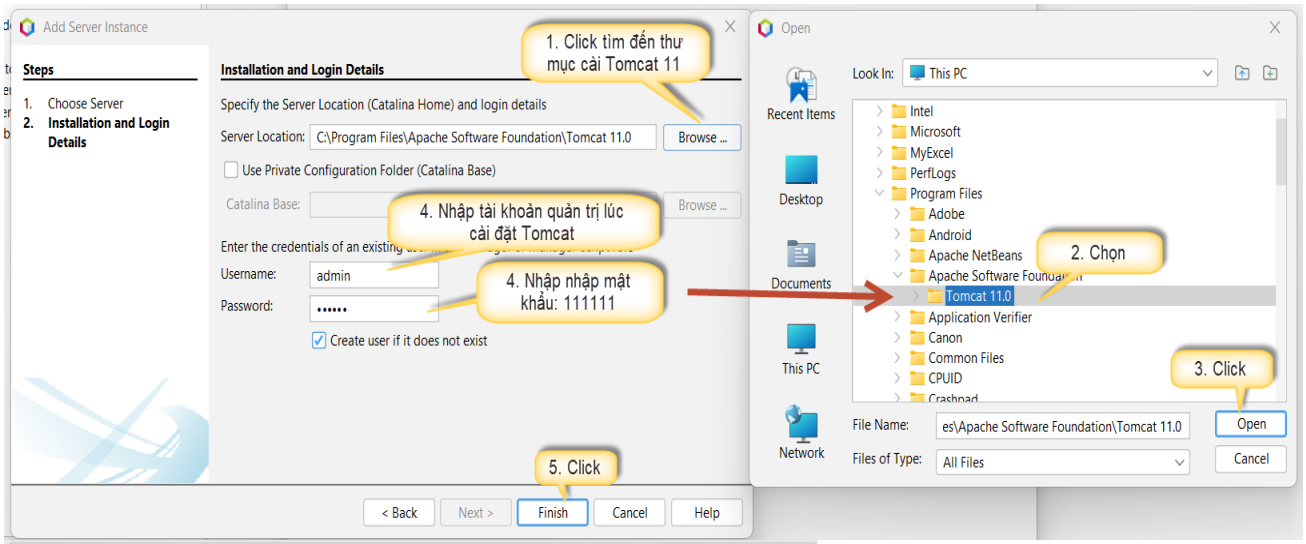


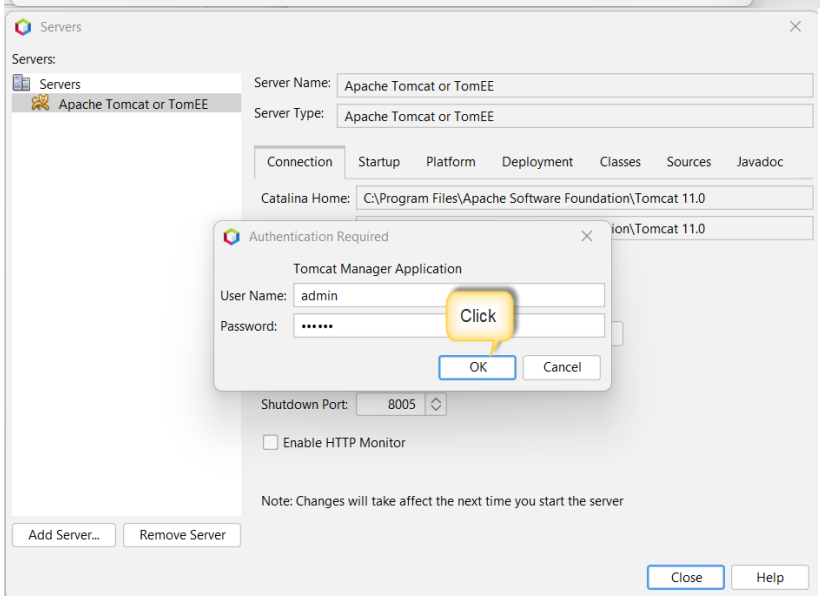
4.4. Tích hợp Apache Tomcat 11 vào NetBeans 21

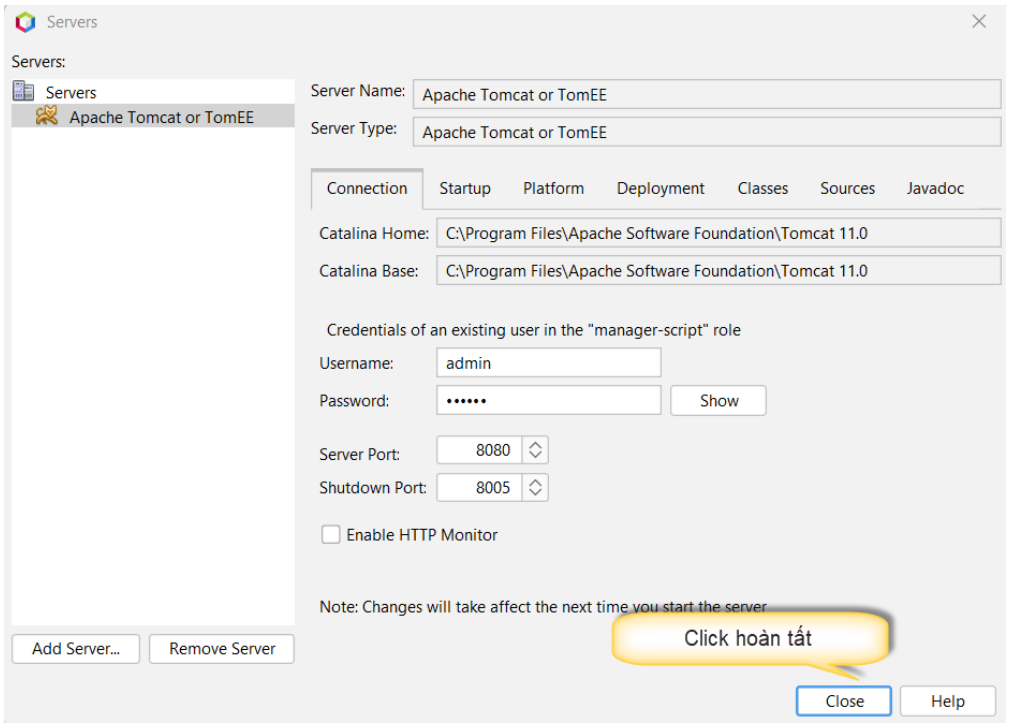
- Khởi động NetBeans 21
- Vào: Tools\Server

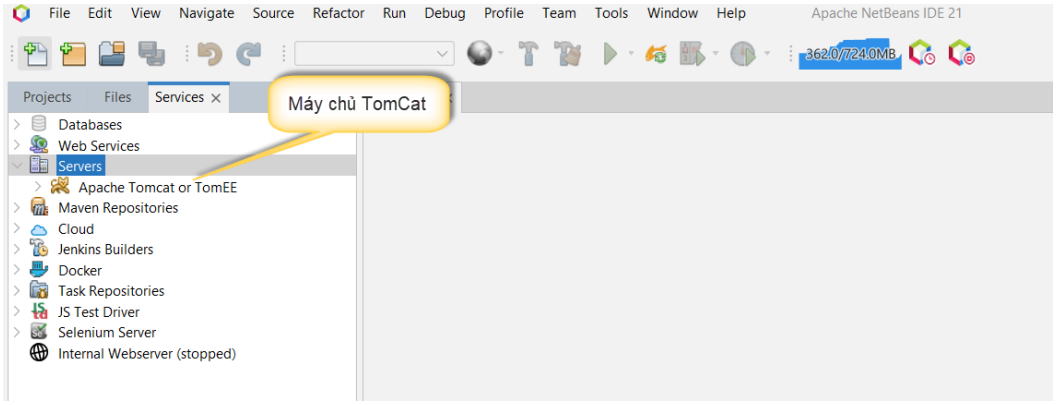





Add Server Instance
Steps
1. Choose Server
2. Installation and Login Details
Installation and Login Details
Specify the Server Location (Catalina Home) and login details
Server Location: C:\Program Files\Apache Software Foundation\Tomcat 11.0
Browse ...
Use Private Configuration Folder (Catalina Base)
Catalina Base:
Browse ...
Enter the credentials of an existing
Username: admin
Password: ••••••
Create user if it does not exist
1. Click tìm đến thư mục cài Tomcat 11
4. Nhập tài khoản quản trị lúc cài đặt Tomcat
4. Nhập nhập mật khẩu: 111111
5. Click
< Back
Next >
Finish
Cancel
Help
Open
Look In: This PC
Recent Items
Desktop
Documents
This PC
Network
Intel
Microsoft
MyExcel
PerfLogs
Program Files
Adobe
Android
Apache NetBeans
Apache Software Foundation
Tomcat 11.0
Application Verifier
Canon
Common Files
CPUID
2. Chọn
3. Click
File Name: es\Apache Software Foundation\Tomcat 11.0
Files of Type: All Files
Open
Cancel


Servers
Servers:
Servers
Apache Tomcat or TomEE
Server Name: Apache Tomcat or TomEE
Server Type: Apache Tomcat or TomEE
Connection
Startup
Platform
Deployment
Classes
Sources
Javadoc
Catalina Home: C:\Program Files\Apache Software Foundation\Tomcat 11.0
Authentication Required
Tomcat Manager Application
User Name: admin
Password: ••••••
Click
OK
Cancel
Shutdown Port: 8005
Enable HTTP Monitor
Note: Changes will take affect the next time you start the server
Add Server...
Remove Server
Close
Help


Servers
Servers:
Servers
Apache Tomcat or TomEE
Server Name: Apache Tomcat or TomEE
Server Type: Apache Tomcat or TomEE
Connection
Startup
Platform
Deployment
Classes
Sources
Javadoc
Catalina Home: C:\Program Files\Apache Software Foundation\Tomcat 11.0
Catalina Base: C:\Program Files\Apache Software Foundation\Tomcat 11.0
Credentials of an existing user in the "manager-script" role
Username: admin
Password: ••••••
Show
Server Port: 8080
Shutdown Port: 8005
Enable HTTP Monitor
Note: Changes will take affect the next time you start the server
Add Server...
Remove Server
Click hoàn tất
Close
Help



4.5. Cài đặt GlassFish5

- Tương tự như Apache Tomcat, GlassFish là một máy chủ ứng dụng (application server) mã nguồn mở dành cho nền tảng Java EE (nay là Jakarta EE). Nó được thiết kế để cung cấp một môi trường hoàn chỉnh và mạnh mẽ để triển khai, quản lý và chạy các ứng dụng doanh nghiệp được xây dựng bằng công nghệ Java.

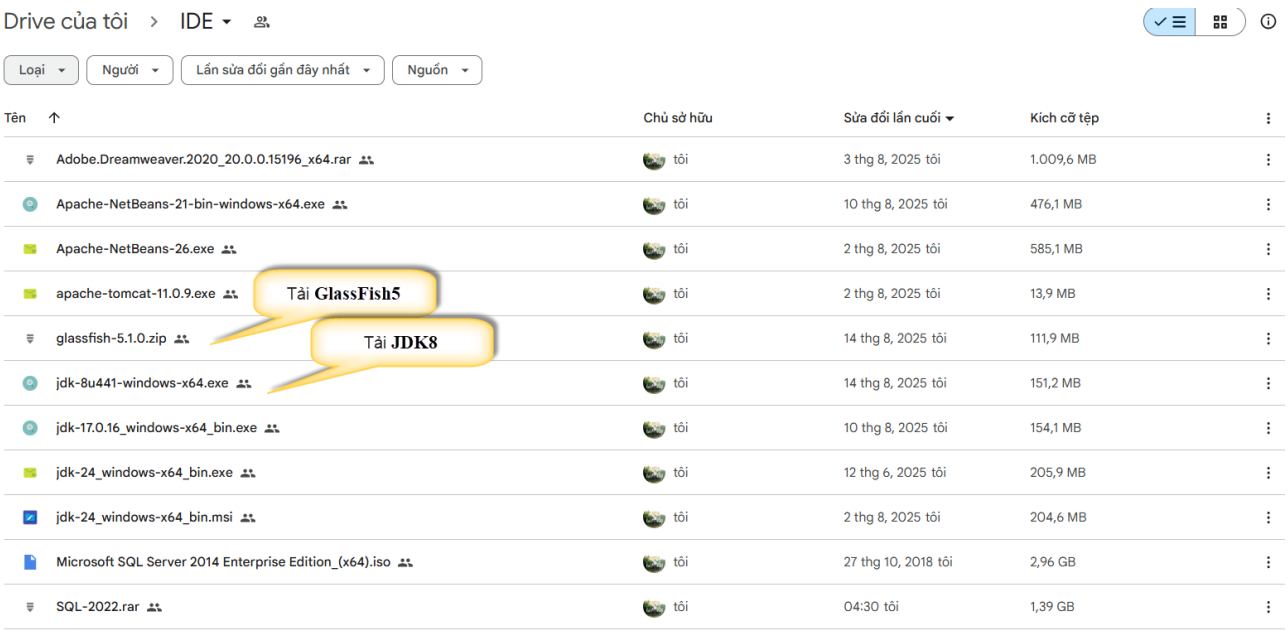
- Nếu cài Apache Tomcat rồi thì không cần cài GlassFish nữa

- Để cài đặt glassfish5, cần phải cài đặt thêm JDK8 trước đó

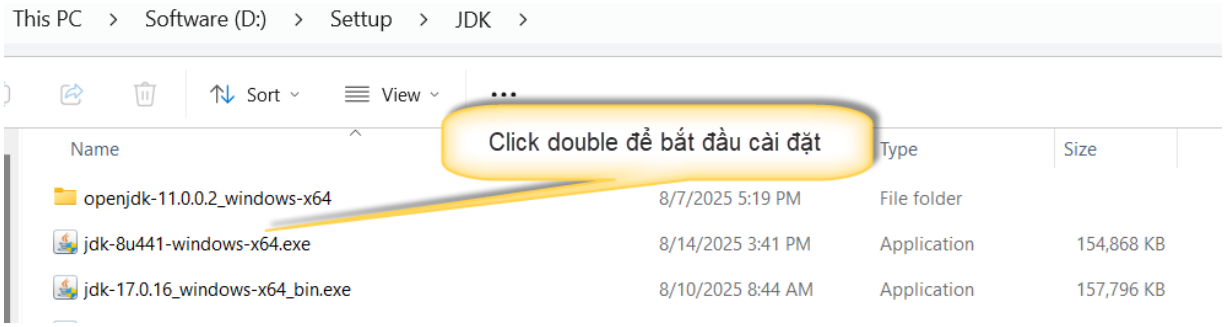
- Sử dụng JDK 8 sẽ đảm bảo sự tương thích hoàn toàn với tất cả các tính năng của GlassFish 5 và các thông số kỹ thuật của Jakarta EE 8. Việc sử dụng các phiên bản JDK mới hơn có thể gây ra lỗi hoặc hành vi không mong muốn.

- Ta có thể dử dụng một trong 2 máy chủ **Tomcat** hoặc **GlassFish** để chạy ứng dụng web

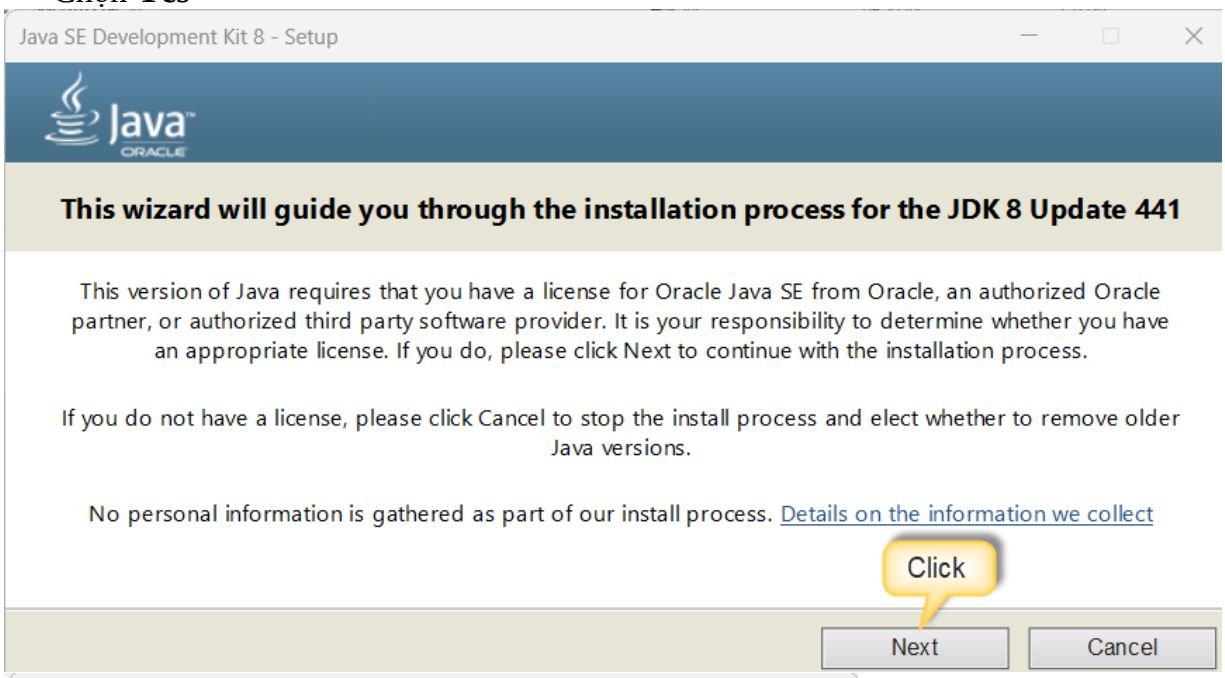
- Tải JDK8 và **GlassFish5** tại địa chỉ: Link

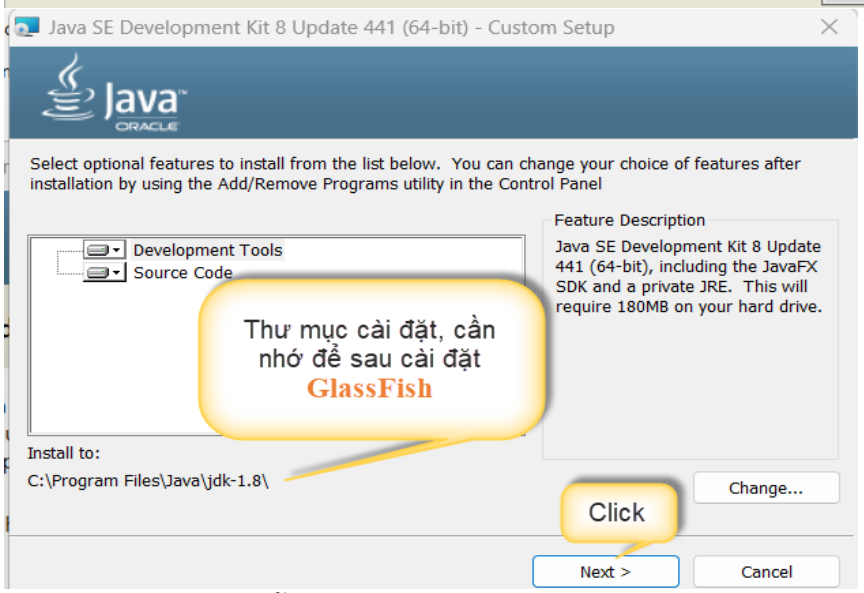


1. Cài JDK8

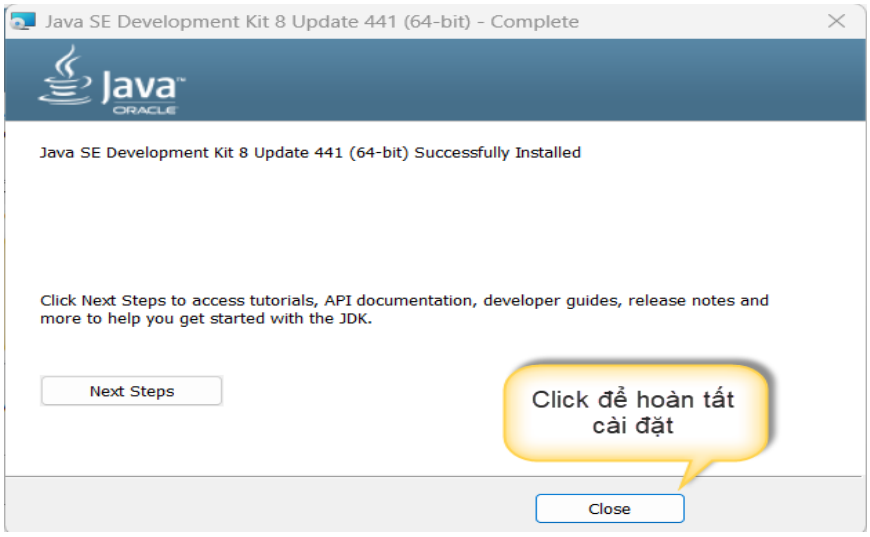


- Chọn **Yes**



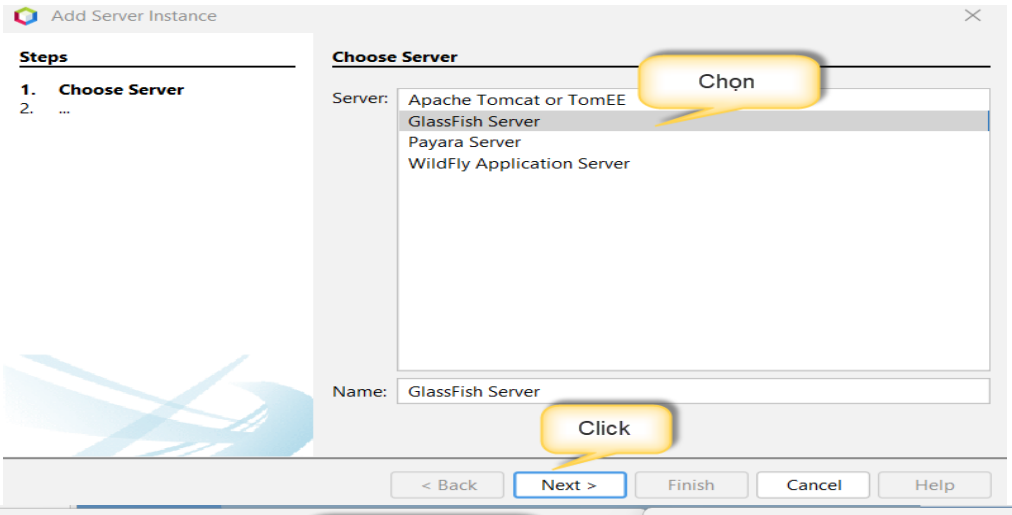


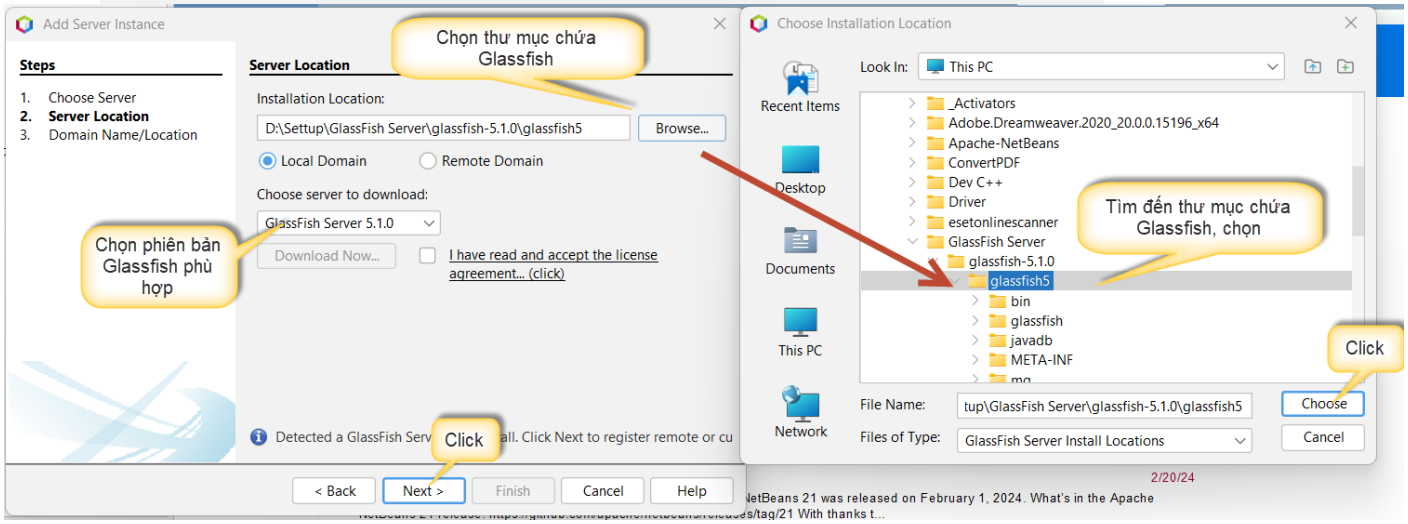
Quá trình cài đặt diễn ra trong ít phút (1-2 phút)

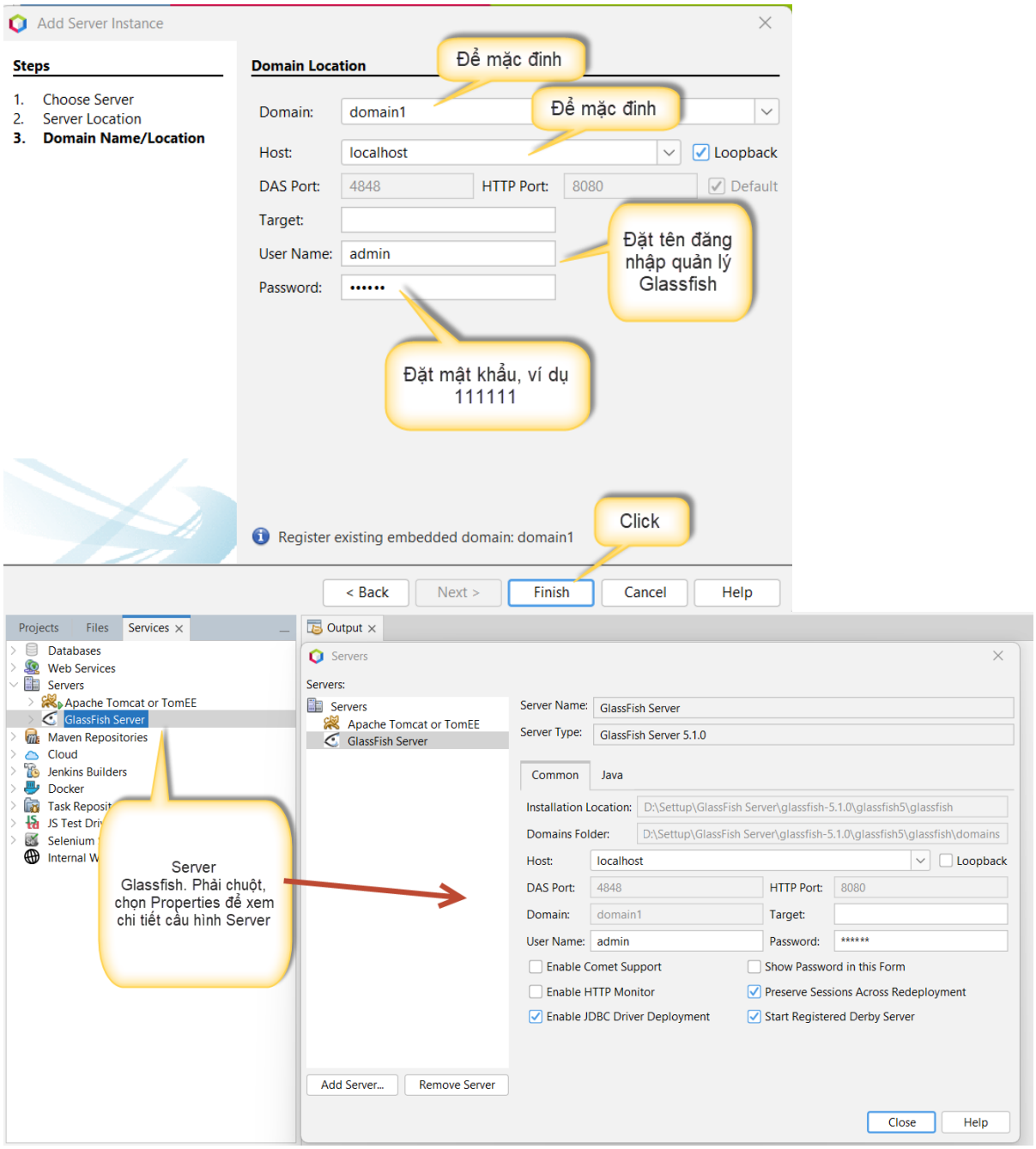


2. Cài GlassFish5

- Khởi động NetBeans 21
- Vào: Tools\Server





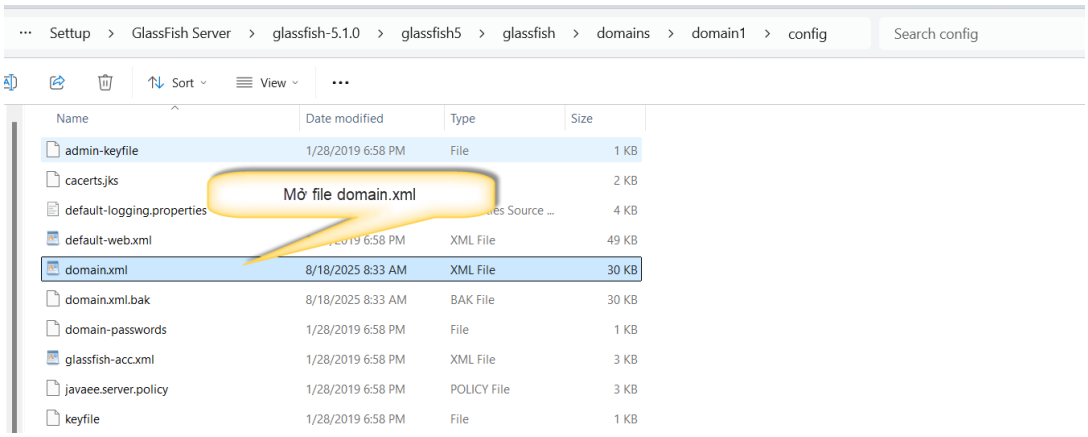


- Nếu chưa cài Tomcat thì cổng mặc định chạy dịch vụ web của Glassfish là 8080 (http://localhost:8080), tuy nhiên cổng này đã được sử dụng bởi Tomcat rồi vì đã đã cài đặt trước đó. Bây giờ ta cần thay đổi cổng này sang 8082 (http://localhost:8082) và cổng quản trị Glassfish từ 4848 thanh 4949(http://localhost:4949), sau đó mới chạy khởi động được Glassfish.

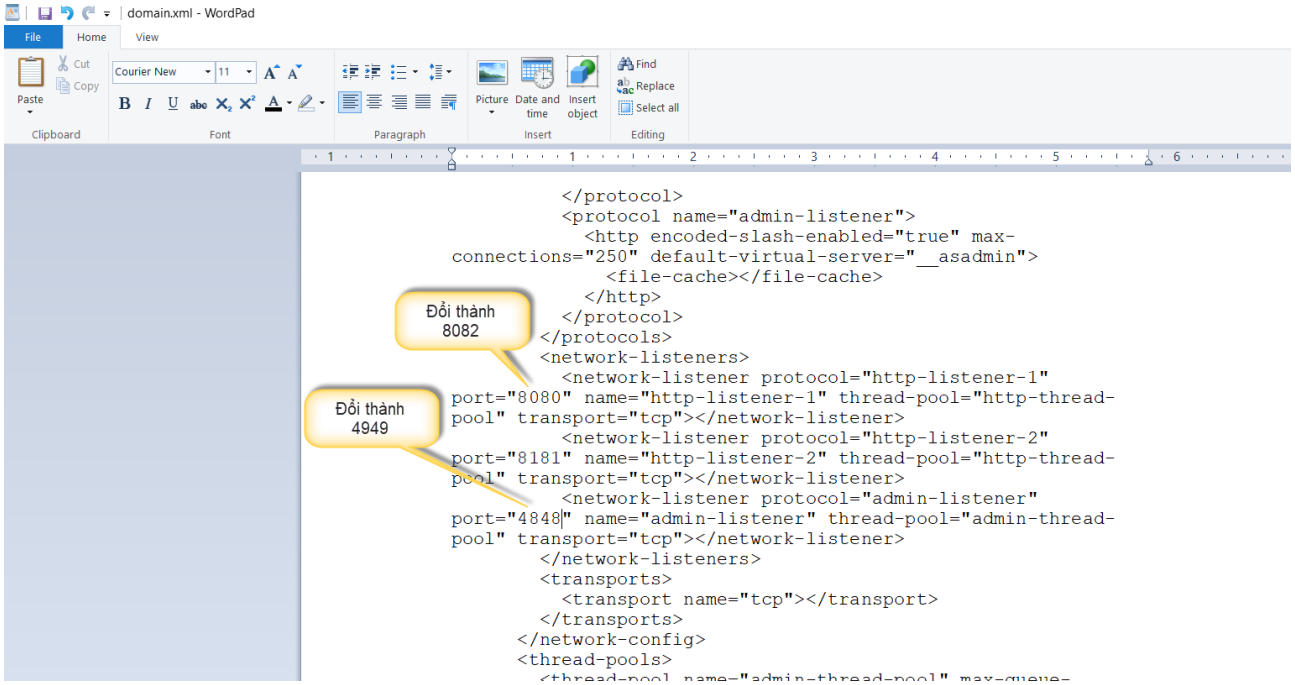
- Ta vào thư mục chưa Glassfish:

Ví dụ: D:\Settup\GlassFish Server\glassfish-5.1.0\glassfish5\glassfish\domains\domain1\config

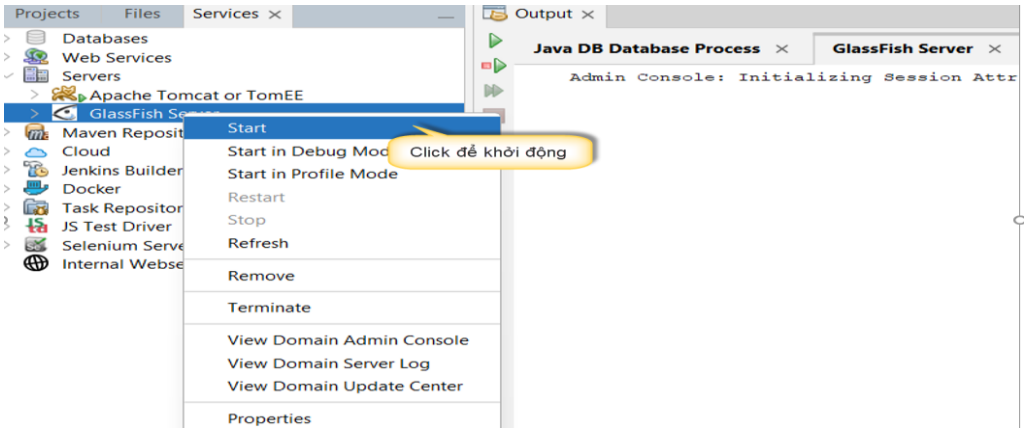
- Mở file domain.xml

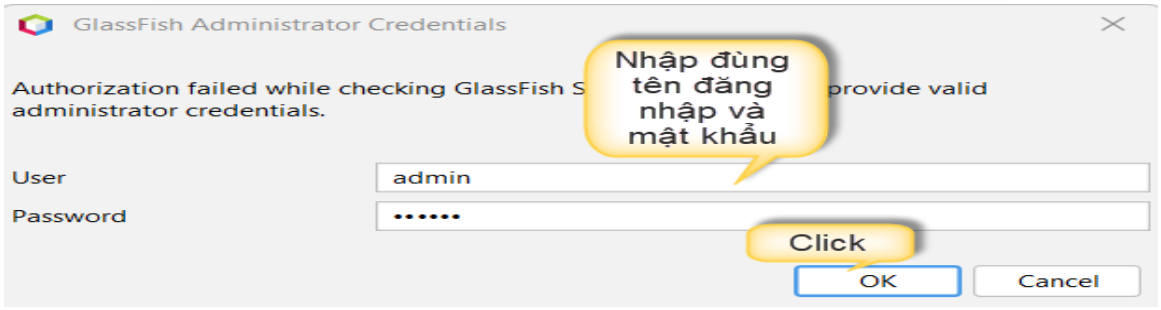


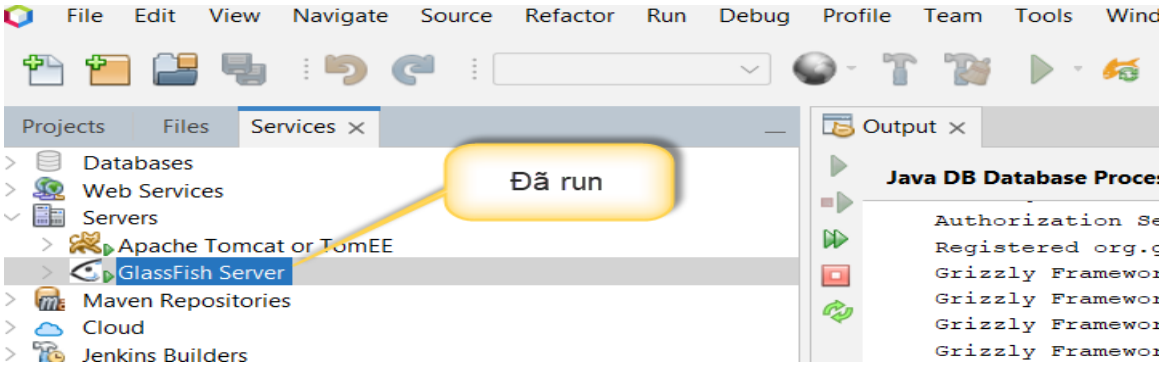
- Tìm đến nội dung **port=”8080”,** Đổi thành **port=”8081”**
- Tìm đến nội dung **port=”4848”,** Đổi thành **port=”4949”**
- Lưu và đóng file



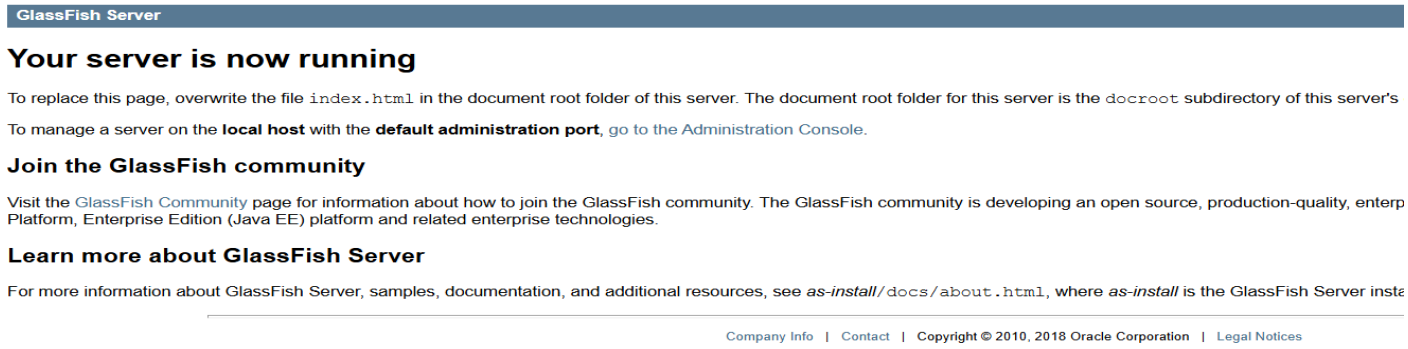
- Khởi động Glassfish trong NetBeans 11
- Mở NetBeans 11, chọn Service, phải chuột vào GlassFish Server, cọn Start



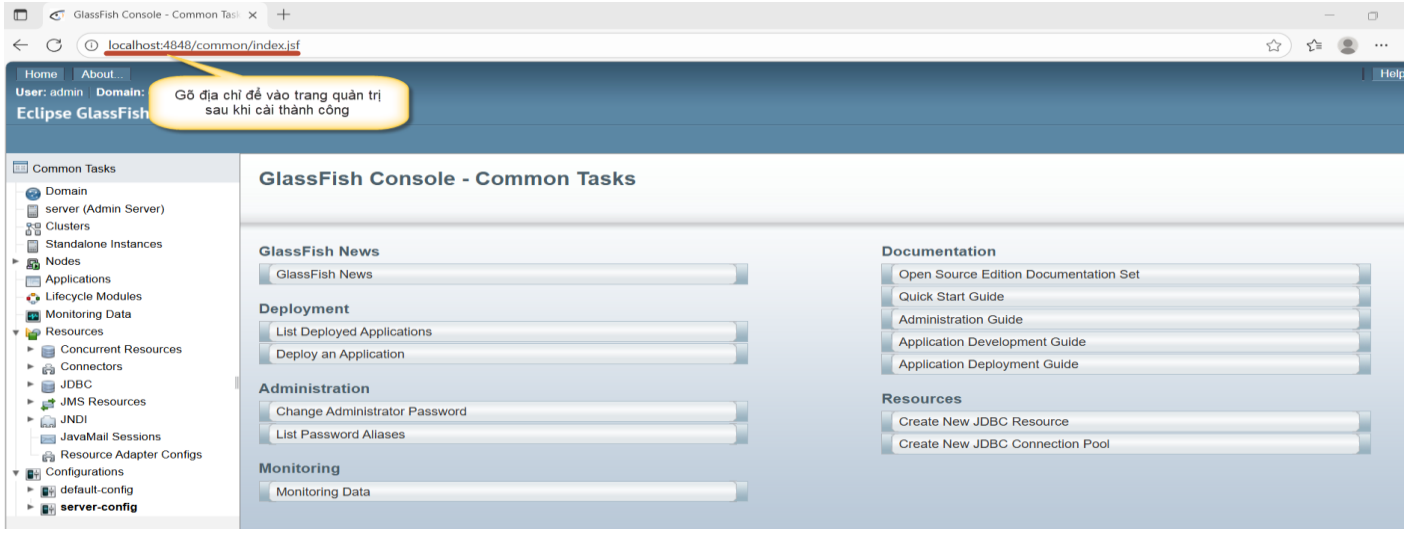




- Như vậy ta đã hoàn tất cài đặt và tích hợp Glassfish vào Netbeans 11
- Địa chỉ server Glassfish chạy dịch vụ web là: http://localhost:8082



- Địa chỉ server Glassfish chạy dịch vụ quản trị là: http://localhost:4949



4.5. Cài đặt SQL Server 2022

- - Xem chi tiết cách thực hiện: Link1 | Link2 (Full)

4.6. Cài đặt Dreamweaver

- - Xem chi tiết cách thực hiện: Link1