

## - 웹 프로그래밍 언어

- Web Browser에서 실행되는 언어를 의미한다.
- 대표적으로 HTML(구조), CSS(표현), JavaScript(동작) 3가지가 있다.
- 이 세 언어를 조합하여 웹 페이지와 웹 애플리케이션을 구현한다.
- 브라우저가 직접 해석할 수 있는 언어는 이 세 가지뿐이며, 서버 측 언어(Java, PHP, Python 등)는 브라우저가 아니라 WAS(Web Application Server)가 처리한다.

## 1. Web FrontEnd

### 1) HTML (HyperText Markup Language)

HTML :





하이퍼텍스트 마크업 언어는 웹 브라우저에 표시되도록 설계된 문서의 표준 마크업 언어입니다. 웹 콘텐츠의 내용과 구조를 정의합니다. 종종 Cascading Style Sheets 및 JavaScript와 같은 스크립팅 언어와 같은 기술의 도움을 받습니다. 위키피디아

개발자 : WHATWG ; World Wide Web Consortium ( W3C ; 이전 명칭)

확장 : SGML

확장 : XHTML

최초 출시 : 1993년; 31년 전

최신 릴리스 : Living Standard

- Hyper Text Markup Language 의 약자
- 웹 브라우저에서 화면을 구성하기 위한 마크업 언어
- 웹 페이지의 GUI (화면 구성) 를 담당
  - 예: 로그인 화면, 회원가입 폼, 게시판 목록, 방명록, 장바구니 화면 등
- 사용자의 입력 값(Form) 을 받거나, 서버에서 전달받은 결과를 화면에 출력하는 역할

- 

## HTML 버전의 발전

- 과거: HTML4
- 현재: HTML5 (표준)
  - HTML5는 단순 화면 구성 뿐 아니라 다양한 웹 API 제공
    - Web Storage (로컬 저장소)
    - Drag & Drop
    - Audio / Video 재생
    - WebSocket (실시간 통신)
    - Canvas / SVG (그래픽 처리)

HTML5 전에는 HTML이 단순 정보를 보여주는 것이었다면

HTML5 이후에 HTML은 웹 애플리케이션 개발을 위한 핵심 기술로 자리 잡음

## HTML 문법 특징

- 대소문자를 구분하지 않음
  - <HTML> 과 <html> 은 동일하게 인식
- 오타가 나면 무시됨
  - 문법 오류가 있어도 프로그램이 중단되지 않고, 해당 부분만 적용되지 않음
- 마크업 언어이므로 태그(< >) 를 이용해 구조를 표현

## 정리

- HTML은 웹 페이지의 구조와 화면을 담당하는 기본 언어
- 현재 표준은 HTML5, 다양한 API 제공
- 문법이 관대하여 오류가 있어도 실행은 되지만, 표준을 지키는 것이 중요

## 2) CSS - Cascading Style sheet

# CSS

프로그래밍 언어 :



## Browser defaults

The browser will style HTML documents using an internal larger than normal text, links are highlighted and structured. Paragraphs are spaced out. List items get a bullet or number.

- Item One
- Item Two

## A level 2 heading

You can change all of this with CSS.

## What is CSS?



더 많은 이미지

캐스케이딩 스타일 시트는 HTML이나 XML과 같은 마크업 언어로 작성된 문서의 표현과 스타일을 지정하는 데 사용되는 스타일 시트 언어입니다. CSS는 HTML과 JavaScript와 함께 월드 와이드 웹의 초석 기술입니다. 위키피디아

파일 이름 확장자 : .css

포함 : HTML 문서

컨테이너 : HTML 요소(태그) 에 대한 스타일 규칙

개발자 : World Wide Web Consortium (W3C)

최초 출시 : 1996년 12월 17일; 27년 전

- Cascading Style Sheets 의 약자
- HTML 문서에 디자인(스타일) 을 적용하는 언어
- 웹 페이지의 레이아웃(Template) 과 시각적 표현을 담당

HTML이 구조(Structure) 를 정의한다면, CSS는 그 구조를 시각적으로 꾸며주는 역할을 한다.

## CSS의 역할

- 텍스트 색상, 크기, 폰트 변경
- 배경 색상 및 이미지 삽입
- 박스 레이아웃(위치, 간격, 정렬) 구성
- 애니메이션 및 트랜지션 효과 적용
- 반응형 웹(Responsive Web) 구현

## 발전과 특징

- 과거에는 플래시(Flash) 나 포토샵 이미지를 활용해 시각적 요소를 구현했으나,

현재는 **CSS3** 만으로도 대부분의 시각적 효과를 표현 가능하다.

- 대소문자를 구분한다.
  - 예: Color ≠ color
- 문법 오류(오타)가 있으면 해당 스타일만 적용되지 않는다.

### 3) JavaScript - HTML에 기능(동작=Event)을 추가

## JavaScript (자바스크립트)

프로그래밍 언어 :







JavaScript는 종종 JS로 약칭되며, HTML과 CSS와 함께 웹의 프로그래밍 언어이자 핵심 기술입니다. 웹사이트의 99%는 웹페이지 동작을 위해 클라이언트 측에서 JavaScript를 사용합니다. 웹 브라우저에는 클라이언트 코드를 실행하는 전용 JavaScript 엔진이 있습니다. 위키피디아

디자이너 : **Brendan Eich**

#### JavaScript란?

- HTML 문서에 기능(동작, 이벤트)을 추가하는 프로그래밍 언어
- 대소문자를 엄격하게 구분한다.
- 객체 기반 언어
  - 객체 = 속성(Property) + 메소드(Method)
  - Java의 객체 개념과 유사하지만, 문법은 다르다.

#### 언어적 특징

- 인터프리터 언어
  - 소스 코드를 한 줄씩 해석하며 실행
  - 오류가 발생하면 순차적으로 출력됨

- **싱글 스레드 기반**

- 브라우저에서 **단일 스레드(Single Thread)** 로 동작
- 동기적 처리(Blocking)가 기본 → 비동기 처리(Ajax, Promise, async/await 등)로 보완 가능

## 👉 결론

웹 브라우저(Chrome, Edge, Firefox, Safari, Opera 등)는 기본적으로 **3가지 언어만 해석할 수 있다.**

1. **HTML** → 구조(Structure)
2. **CSS** → 표현(Style)
3. **JavaScript** → 동작(Behavior)

## HTML vs CSS vs JavaScript 비교표

구분	HTML	CSS	JavaScript	
역할	웹 페이지의 <b>구조</b> 정의	웹 페이지의 <b>디자인/스타일</b> 적용	웹 페이지의 <b>동작/기능</b> 구현	
언어 유형	마크업 언어	스타일 언어	프로그래밍 언어	
예시	제목, 본문, 폼 등 화면 요소 구성	색상, 폰트, 레이아웃, 애니메이션	버튼 클릭, 입력값 검증, 동적 데이터 처리	
버전	HTML5 (최신 표준)	CSS3 (최신 표준)	ECMAScript(ES6+ 최신 문법)	
대소문자	구분하지 않음	구분함	구분함	
실행 환경	웹 브라우저	웹 브라우저	웹 브라우저 (싱글 스레드)	

## Web Front 언어의 한계와 단점

### 1. 소스 코드 공개

- HTML, CSS, JavaScript는 클라이언트(브라우저) 측에서 실행되므로, 개발자가 작성한 소스 코드가 그대로 노출된다.
- 따라서 보안에 취약하며, 로직이나 알고리즘을 보호하기 어렵다.

### 2. 클라이언트 사이드 언어의 한계

- 웹 브라우저에서 실행되므로, 브라우저 종류와 버전에 따라 해석 결과가 다를 수 있다.
- 같은 코드라도 실행 결과가 달라지는 **Cross-Browser 이슈**가 존재한다.

- 이러한 문제를 해결하기 위해 JavaScript 기반의 다양한 라이브러리와 프레임워크가 등장하였다.

### 3. DB 연동 불가

- JavaScript 자체만으로는 데이터베이스에 직접 접근할 수 없다.
- 즉, 영속성(persistence)을 보장하지 못하며, 데이터를 저장하고 관리하려면 반드시 **Back-End 서버(예: Java, Node.js, Python, PHP 등)**가 필요하다.

### 4. 데이터 처리 한계

- HTML Form을 통해 서버로 데이터를 전송할 수는 있지만, 전송 후 응답을 받는 과정에서 클라이언트만으로 데이터를 유지하거나 조작하는 데 한계가 있다.
- 따라서 **서버 측 언어와의 협력이 필수적이다.**

### 5. 상태 유지 불가 (Stateless 특성)

- 웹은 기본적으로 **Stateless 프로토콜(HTTP)** 기반이므로, 페이지 이동 시마다 새로운 요청(Request)과 응답(Response)이 발생한다.
- 이 과정에서 상태정보(Session, 로그인 상태, 장바구니 등)를 유지하기 위해서는 별도의 기술(쿠키, 세션, 토큰 등)이 필요하다.

### 6. 보안 취약성

- 입력 값 검증(Validation)을 클라이언트 단에서만 처리하면 조작이 가능하다.
- 따라서 중요한 보안 로직은 반드시 Back-End에서 처리해야 한다.

## jQuery의 등장과 현대 JS 프레임워크로의 발전 흐름

### • 과거(2000년대 중반 이전)

- JavaScript는 표준화가 미흡했고, 브라우저별 동작 방식 차이 때문에 개발자들이 크로스 브라우저 문제로 큰 불편을 겪었다.
- DOM 조작, 이벤트 처리, Ajax 호출도 브라우저마다 코드가 달라야 했다.

### • jQuery의 등장 (2006)

- **"Write less, Do more"** 라는 슬로건과 함께 단순화된 API 제공.
- 복잡했던 DOM 조작, 이벤트 바인딩, Ajax 통신을 간단한 코드로 구현할 수 있었고, **브라우저 호환성 문제를 획기적으로 줄였다.**
- 웹 개발자들이 가장 많이 사용하는 필수 라이브러리로 자리 잡았다.

### • 현대의 변화 (2010년대 이후)

- 웹 애플리케이션이 점점 복잡해지면서, 단순히 DOM 조작 중심의 jQuery만으로는

한계가 생겼다.

- 대규모 프로젝트에서는 **상태 관리, 컴포넌트 기반 개발, 성능 최적화**가 필요해졌다.
- 이 흐름 속에서 **Angular, React, Vue.js** 같은 **JavaScript 프레임워크/라이브러리**가 등장.
  - **Angular** : 구글이 만든 풀스택 프레임워크, 대규모 엔터프라이즈 개발에 적합.
  - **React** : 컴포넌트 기반 UI, 가상 DOM을 통한 성능 최적화.
  - **Vue.js** : 진입 장벽이 낮고 직관적이며, 양방향 바인딩 지원.
- 이들은 단순한 DOM 제어가 아니라 **애플리케이션 아키텍처를 구축**할 수 있도록 지원한다.

## • 오늘날

- jQuery는 여전히 간단한 프로젝트나 레거시 코드 유지보수에서 사용되지만,
- 새로운 프로젝트에서는 대부분 **현대적인 JS Framework/Library**가 선택된다.
- 즉, 웹 개발은 **단순한 웹 페이지 제작** → **웹 애플리케이션 개발**로 확장되었고, 이에 따라 프론트엔드 기술도 성숙해졌다.

## 현대 Web Front-End 개발과 JavaScript 생태계

### 1. MEAN Stack 개발자

MEAN 스택은 **JavaScript 기반 풀스택 개발 환경**으로, 프론트엔드부터 백엔드, 데이터베이스까지 모두 JavaScript로 통일하여 개발 효율을 극대화한다.

- **MongoDB** : NoSQL 기반(비관계형 데이터베이스)으로 JSON 형태의 문서(Document)를 저장하여 유연한 데이터 모델을 지원한다.
- **Express.js** : Node.js 위에서 동작하는 서버 프레임워크로, 요청과 응답을 중간에서 제어하는 **Controller 역할**을 담당한다.
- **Angular.js** : 프론트엔드 View 역할을 담당하는 프레임워크로, MVC/MVVM 패턴을 적용할 수 있다.
- **Node.js** : 구글 V8 엔진 위에서 동작하는 자바스크립트 런타임으로, 서버 사이드 로직을 처리할 수 있도록 한다.

즉, MEAN 스택은 **JavaScript 하나의 언어로 풀스택 개발**이 가능한 환경을 제공한다는 점에서 큰 장점을 가진다.

### 2. JavaScript 기반 프레임워크의 등장

초창기에는 단순히 DOM 조작과 이벤트 처리 중심의 **라이브러리(jQuery 등)**가 주로 사용되었다. 그러나 웹 애플리케이션이 복잡해짐에 따라, 구조적 개발을 위한 **프레임워크**가 필요하게 되었고, **MVVM(Model-View-ViewModel)** 아키텍처를 도입한 다양한 프레임워크가 등장하였다.

### 3. 주요 프론트엔드 프레임워크

#### (1) Angular.js

- 구글과 오픈 커뮤니티에서 개발한 풀스택형 프레임워크.
  - **장점**
    - 데이터 바인딩, 폼 처리, 라우팅, 상태 관리 등 **완전한 기능 세트** 제공
    - TypeScript 기반 → 정적 타입 검사로 코드 안정성 확보
  - **단점**
    - 학습 난이도가 높고 구조가 복잡
    - 프레임워크 자체 크기가 크며 초기 로딩 속도가 느림
    - 업데이트 주기가 빨라 코드 호환성 문제 발생 가능
- 대규모 엔터프라이즈급 프로젝트에 적합하다.

#### (2) React.js

- 페이스북(메타)에서 개발한 **UI 라이브러리**.
- **특징**
  - 가상 DOM(Virtual DOM) 활용 → 빠르고 효율적인 UI 업데이트
  - 컴포넌트 기반 개발 → 재사용성과 유지보수성 강화
  - 대규모 커뮤니티와 생태계 보유 → 다양한 라이브러리와 도구 지원
- **한계**
  - 기본적으로 View에만 초점 → 라우팅, 상태 관리 등은 추가 라이브러리 필요
  - 학습곡선이 존재 (JSX, 상태 관리 패턴 등)

유연성과 확장성이 뛰어나 스타트업부터 대규모 서비스까지 폭넓게 사용된다.

#### (3) Vue.js

- 구글 출신 개발자 **Evan You**가 개인 프로젝트로 시작하여 발전한 프레임워크.
- **장점**
  - 가볍고 빠르며, 직관적이고 개발자 친화적인 문법
  - 진입 장벽이 낮아 빠른 학습 가능



- 단점

- React, Angular에 비해 생태계가 작아 대규모 프로젝트 지원 도구가 다소 부족  
소규모~중규모 프로젝트나 빠른 프로토타입 개발에 강점이 있다.

#### 4. 그 외 주요 기술

- **Next.js** : React 기반 프레임워크, 서버사이드 렌더링(SSR)과 정적 사이트 생성(SSG)을 지원
- **Nuxt.js** : Vue 기반 프레임워크, SEO 최적화와 SSR 제공
- **Node.js** : 자바스크립트를 서버 환경에서 실행할 수 있는 런타임 (서버사이드 스크립트 언어가 아니라 실행 환경임)
- **Vanilla JS** : 프레임워크/라이브러리 없이 사용하는 순수 자바스크립트

#### 5. 자바스크립트 개발을 보조하는 핵심 도구

- **TypeScript** : Microsoft에서 개발한 JS 확장 언어로, 타입 시스템을 도입하여 대규모 개발의 안정성을 높임.
- **Babel** : 최신 JS 문법(ES6+)을 구 버전 브라우저에서도 실행 가능하도록 변환해주는 컴파일러.
- **Webpack** : 모듈 번들러. HTML, CSS, JS, 이미지 등을 모듈 단위로 관리하고 최적화하여 하나의 결과물로 번들링.
- **ESLint** : 코드 분석 도구. 잘못된 문법이나 스타일을 자동 검출해 코드 품질을 향상.

### Web Front → Web Back-End 기술의 필요성

#### 1. Web Front 언어의 한계

앞서 살펴본 것처럼, HTML, CSS, JavaScript 는 웹 페이지 제작에는 유용하지만,

- DB 연동 불가
  - 상태 정보 유지 불가
  - 보안 취약
- 등의 단점이 존재한다.

따라서 실제 서비스 가능한 동적 웹 애플리케이션(Dynamic Web Application) 을 만들기 위해서는 **Web Back-End 기술**이 반드시 필요하다.

## 2. 웹 프로젝트 유형

- **Static Web Application**
  - HTML, CSS, JavaScript 만으로 구성된 정적 웹
  - 단순 정보 제공 사이트(예: 홍보 페이지, 소개 페이지)
- **Dynamic Web Application**
  - Front-End + Back-End 언어가 결합된 웹
  - DB 연동, 사용자 맞춤 서비스 제공 가능
  - 예: 쇼핑몰, SNS, 온라인 banking 서비스

## 3. Web Back-End 기술

대표적인 Back-End 기술에는 **Servlet & JSP (Java 기반)**, **ASP (Microsoft)**, **PHP (오픈소스)** 등이 있다.

우리 과정에서는 **Java 기반 Servlet & JSP** 를 학습한다.

## 4. Java 와 웹 개발 영역

- **Java SE (Standard Edition)** : 일반 응용프로그램 개발
- **Java EE (Enterprise Edition)** : 웹/엔터프라이즈 애플리케이션 개발
- **Java ME (Micro Edition)** : 모바일 환경용 애플리케이션 개발

## 5. Servlet & JSP 구조 (MVC 패턴 적용)

- **JSP** : HTML, CSS, JavaScript 작성 → **View(화면)** 역할
- **Servlet** : 요청/응답 처리 → **Controller** 역할
- **Java SE 클래스 (Service, DAO, DTO/VO 등)** → **Model** 역할

결론적으로, 웹 프로젝트는 **MVC(Model-View-Controller)** 구조로 구축한다.

DB 연동을 통해 데이터의 **영속성(persistence)** 을 유지하며, 이를 위해 **JDBC(Java Database Connectivity)** 를 활용한다.

## 6. Back-End 코드의 실행 과정

- 브라우저는 **HTML, CSS, JavaScript** 만 해석할 수 있다.
- 따라서 **Servlet & JSP** 는 브라우저가 직접 해석하지 못한다.
- 이를 해결하기 위해 **WAS(Web Application Server)** 가 필요하다.
- WAS 는 **Servlet/JSP** 코드를 해석하여 **HTML** 로 변환한 뒤 브라우저에 전달한다.

## 7. Servlet Container (WAS)

- **Servlet Container = WAS(Web Application Server)**
- Java Servlet & JSP 실행 환경 제공
- 가장 대표적인 WAS : **Apache Tomcat**
  - Web Server + Servlet Container 기능 제공
  - 무료로 사용 가능 (<https://tomcat.apache.org/>)

## 8. 주요 Web Application Server 종류

1. BEA Systems → **WebLogic**
2. IBM → **WebSphere**
3. TMAX → **Jeus**
4. RedHat → **JBoss**
5. Sun Microsystems → **Sun Application Server**
6. Apache Project → **Tomcat**

## Ajax 와 데이터 포맷(XML → JSON)의 이해

### 1. Ajax 의 등장 배경

웹 초창기에는 서버와의 통신이 단순했다.

- 사용자가 버튼을 클릭하거나 데이터를 입력하면 **새로운 요청(Request)** 을 보내야 했고,
- 서버는 전체 페이지를 다시 그려서 **응답(Response)** 을 보냈다.

결과적으로 페이지 전체가 새로고침 되면서 느린 속도와 불편한 사용자 경험이 발생했다.

#### 문제의식

“전체 페이지를 새로 고치지 않고, 필요한 부분만 바꿀 수는 없을까?”

### 2. Ajax 의 개념과 원리

- **Ajax (Asynchronous JavaScript And XML)**
  - 비동기 통신 기술
  - **JavaScript + XMLHttpRequest 객체**를 활용하여 서버와 통신
  - 전체 페이지를 새로고침하지 않고, **일부분만 갱신(Update)** 가능

오늘날에는 **fetch() API**, **jQuery.ajax()**, **axios** 등 다양한 방식으로 구현 가능하다.

### 3. Ajax 가 바꾼 웹 개발 패러다임

- **Before Ajax** : 요청 시마다 전체 페이지 Reload
- **After Ajax** : 필요한 데이터만 서버에서 받아와 페이지 일부만 동적 갱신

이를 통해 웹은 단순 "문서 뷰어"에서 벗어나, **데스크톱 애플리케이션 수준의 사용자 경험(UX)** 을 제공할 수 있게 되었다.

Google Maps, Gmail 같은 서비스가 Ajax 혁신의 대표 사례.

### 4. Ajax 와 데이터 포맷 - XML 의 한계

Ajax 초기에는 서버와 데이터를 교환할 때 주로 **XML** 을 사용했다.

- **XML (Extensible Markup Language)**
  - 개발자가 태그를 직접 정의할 수 있음
  - 시스템 간 데이터 교환 포맷으로 많이 사용됨
  - 예:

```
<boards>
  <board>
    <no>1</no>
    <subject>제목</subject>
    <content>내용</content>
  </board>
</boards>
```

- **XML 의 단점**
  - 문법이 엄격하고 파일 크기가 무겁다.
  - 데이터 해석(Parsing) 과정이 복잡하다.
  - 같은 데이터를 표현해도 불필요한 태그 때문에 **비효율적**.

따라서 점차 **JSON(JavaScript Object Notation)** 으로 대체되었다.

### 5. JSON 의 부상

- **JSON (JavaScript Object Notation)**
  - 자바스크립트 객체 표기법을 기반으로 한 경량 데이터 포맷
  - 가볍고 직관적이며, JavaScript 와 호환성이 높다.
  - XML 보다 훨씬 간단하게 구조화 가능

- 예:

```
[
  {"no":1, "subject":"제목", "content":"내용"},
  {"no":2, "subject":"제목", "content":"내용"},
  {"no":3, "subject":"제목", "content":"내용"}
]
```

- 장점

- 가볍고 네트워크 전송 효율 높다.
- 파싱이 간단 (JavaScript 에서 JSON.parse() 로 바로 사용 가능)
- 가독성이 높고 직관적

현재 웹 개발에서는 **Ajax + JSON** 조합이 사실상 표준으로 자리잡았다.

## 6. XML 의 현재 위치

XML 은 완전히 사라진 것은 아니다.

- 여전히 **설정 파일(Configuration)** 용도로 많이 사용됨
  - 예: web.xml, server.xml 등
- 데이터 교환보다는 **메타데이터 정의나 환경 설정**에 적합한 포맷으로 자리잡음.

## 결론

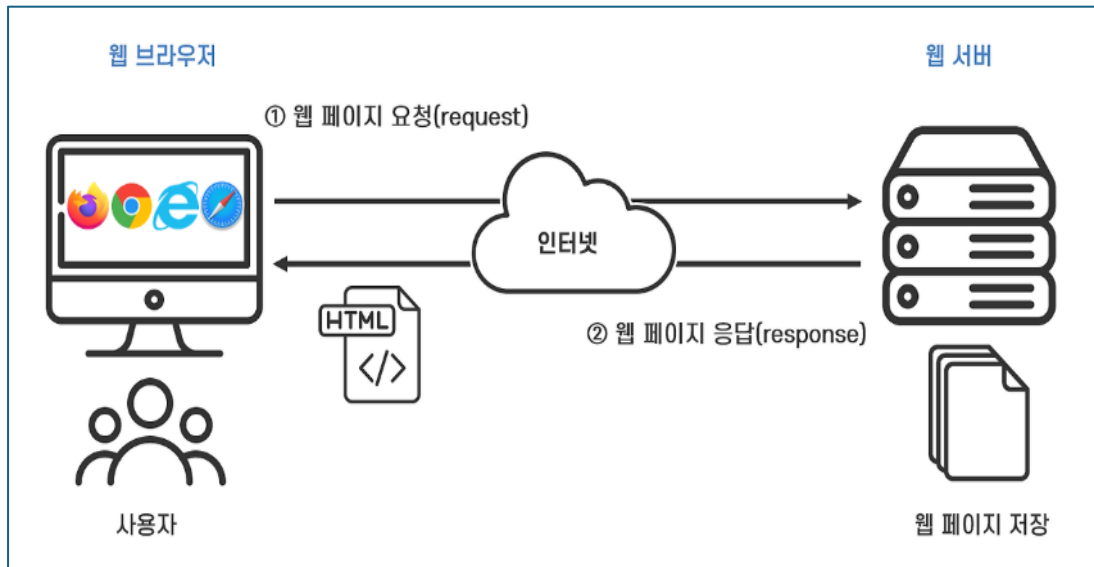
- **Ajax** 는 웹 페이지 전체 새로고침 없이 서버와 비동기 통신을 가능하게 한 혁신적 기술이다.
- **XML** 은 초기에 Ajax 데이터 포맷으로 사용되었지만, 무겁고 불편하다는 한계가 있었다.
- **JSON** 은 가볍고 직관적인 데이터 포맷으로, 오늘날 대부분의 웹 애플리케이션에서 표준으로 사용된다.

따라서 현대 웹 개발에서는 **Ajax (fetch, axios 등) + JSON** 조합이 기본 패턴으로 활용된다.

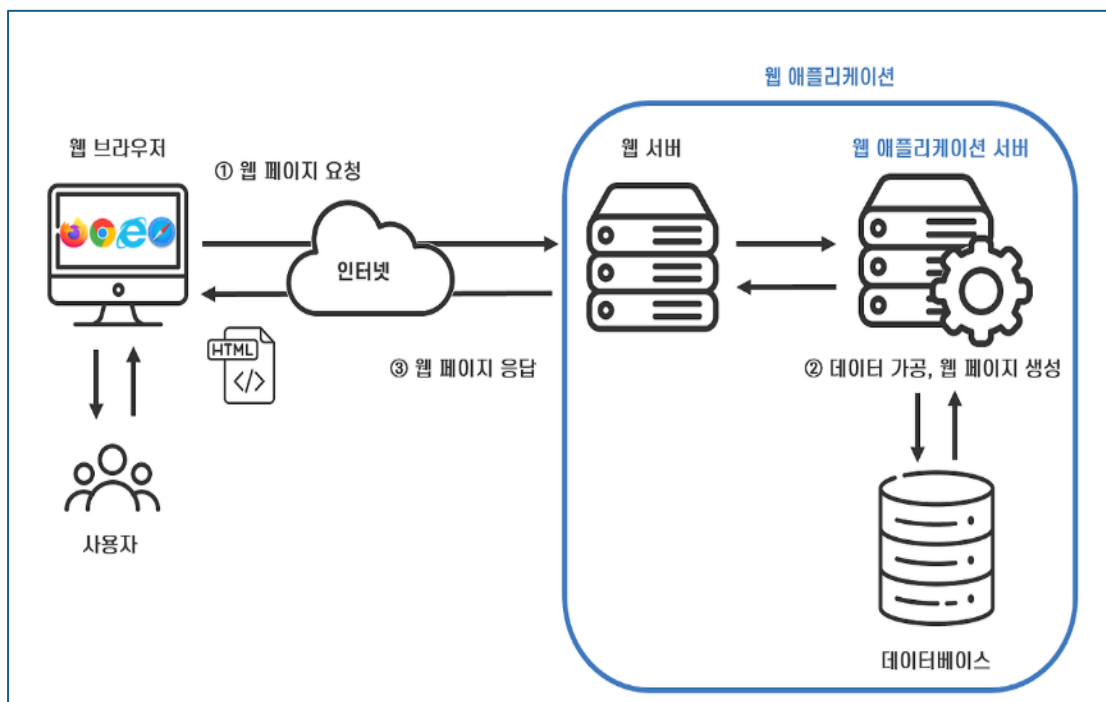
## HTTP(HyperText Transfer Protocol)란?

: 서버와 클라이언트 서로 데이터를 주고 받기 위해 사용되는 **통신규약**이다.

## &lt;웹 브라우저와 웹 서버&gt;



## &lt;웹 애플리케이션&gt;



## Request (요청)

클라이언트가 서버에게 연락하는 것을 요청이라고 하며 요청을 보낼때는 요청에 대한 정보를 담아 서버로 보낸다.

Request Method (요청의 종류)

GET : 자료를 요청할 때 사용  
 POST : 자료의 생성을 요청할 때 사용  
 PUT : 자료의 수정을 요청할 때 사용  
 DELETE : 자료의 삭제를 요청할 때 사용

#### ☞ Response (응답)

서버가 요청에 대한 답변을 클라이언트에게 보내는 것을 응답이라고 한다.

Status Code (상태 코드)

상태 코드에는 굉장히 많은 종류가 있다.

모두 숫자 세 자리로 이루어져 있으며, 아래와 같이 크게 다섯 부류로 나눌 수 있다.

- ✓ 1XX (조건부 응답) : 요청을 받았으며 작업을 계속한다.
- ✓ 2XX (성공) : 클라이언트가 요청한 동작을 수신하여 이해했고 승낙했으며 성공적으로 처리했음을 가리킨다.
- ✓ 3XX (리다이렉션 완료) : 클라이언트는 요청을 마치기 위해 추가 동작을 취해야 한다.
- ✓ 4XX (요청 오류) : 클라이언트에 오류가 있음을 나타낸다.
- ✓ 5XX (서버 오류) : 서버가 유효한 요청을 명백하게 수행하지 못했음을 나타낸다.

**참고) 아래 링크를 클릭해서 아래의 그림 항목을 찾아서 읽어보세요.**

<https://developer.mozilla.org/ko/docs/Web/HTTP/Overview>


### HTTP은 상태는 없지만 세션은 있습니다

HTTP는 상태를 저장하지 않습니다(Stateless). 동일한 연결 상에서 연속하여 전달된 두 개의 요청 사이에는 연결고리가 없습니다. 이는 e-커머스 쇼핑 바구니처럼, 일관된 방식으로 사용자가 페이지와 상호작용하길 원할 때 문제가 됩니다. 하지만, HTTP의 핵심은 상태가 없는 것이지만 HTTP 쿠키는 상태가 있는 세션을 만들도록 해줍니다. 헤더 확장성을 사용하여, 동일한 컨텍스트 또는 동일한 상태를 공유하기 위해 각각의 요청들에 세션을 만들도록 HTTP 쿠키가 추가됩니다.

## HTTP 흐름

클라이언트가 서버와 통신하고자 할 때, 최종 서버가 됐든 중간 프록시가 됐든 다음 단계의 과정을 수행합니다.

1. TCP 연결을 엽니다. TCP 연결은 요청을 보내거나(혹은 여러 개의 요청) 응답을 받는데 사용됩니다. 클라이언트는 새 연결을 열거나, 기존 연결을 재사용하거나, 서버에 대한 여러 TCP 연결을 열 수 있습니다.
2. HTTP 메시지를 전송합니다. HTTP 메시지(HTTP/2 이전)는 인간이 읽을 수 있습니다. HTTP/2에서는 이런 간단한 메시지가 프레임 속으로 캡슐화되어 직접 읽는게 불가능하지만 원칙은 동일합니다.

```
HTTP 

GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

3. 서버에 의해 전송된 응답을 읽어들이니다

## URI(Uniform Resource Identifier) vs URL (Uniform Resource Locator)

### 1) URI

Uniform 은 리소스를 식별하는 통일된 방식이고 Resource URI 로 식별이 가능한 모든 종류의 자원(웹 브라우저 파일 및 그 이외의 리소스 포함)을 지칭한다.

Identifier 는 다른 항목과 구분하기 위해 필요한 정보를 뜻한다.

즉, URI 는 인터넷상의 리소스 "자원 자체"를 식별하는 고유한 문자열 시퀀스이다.

자원의 위치 뿐만 아니라 자원에 대한 고유 식별자로서 URL 을 포함한다.

### 2) URL

Uniform Resource Locator, 네트워크상에서 통합 자원(리소스)의 "위치"를 나타내기 위한 규약으로 자원 식별자와 위치를 동시에 나타낸다. 웹 사이트 주소 + 컴퓨터 네트워크 상의 자원, 이는 웹 사이트 주소 뿐만 아니라 컴퓨터 네트워크 상의 자원을 모두 나타내는 표기법이다.

자원이 실제로 존재하는 위치를 가리킨다.

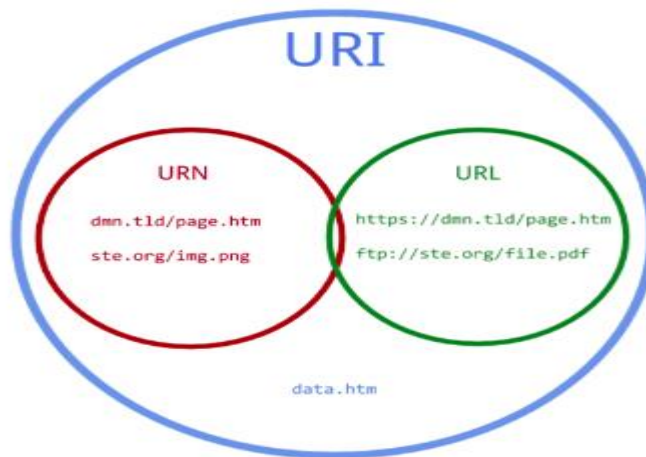
### 3) URN ( Uniform Resource Name ) - 통합 자원 이름

Resource 의 위치와 상관없이 식별 가능한 고유한 이름 역할

URN 은 이름이 변하지 않는한, 리소스 위치가 변경되더라도 문제없이 동작한다.



즉, 리소스 위치와 상관없이 이름만으로 식별할 수 있다는 개념이다.



URI 는 식별하고, URL은 위치를 가르킨다.

## SPA(Single Page Application)

웹 사이트의 전체 페이지를 하나의 페이지에 담아 동적으로 화면을 변경해 표시하는 기술  
브라우저는 최초에 한번 서버에 요청하여 페이지 전체를 로드하고 이후부터는 특정 변경 사항이 일어나는 부분만 Ajax 와 같은 기술을 통해 데이터를 바인딩하는 방식으로 동작(사용자에게 빠른 인터랙션 제공)

## SSR(Server Side Rendering ) vs CSR(Client Side Rendering) 개념

### 1) SSR

서버 사이드 렌더링이라는 뜻으로 구동 방식은 서버에서 렌더링하여 완성된 HTML 파일을 로드해 준다. 클라이언트에서 요청할 때마다 각 상황에 맞는 HTML 파일을 넘겨주기 때문에 페이지가 여러 개 이므로 **MPA(Multi Page Application) 에 적합한 환경**

### 2) CSR

클라이언트 사이드 렌더링이라는 뜻으로 구동 방식은 초기 로드 시 빈 HTML 과 모든 로직이 담겨 있는 Javascript 다운로드를 하고 그 후 빈 HTML 에 Javascript 를 이용하여 Dom 을 동적으로 생성하여 그린다. - **SPA 에 적합한 환경**

\* SSR 는 검색엔진 최적화가 가능하지만 CSR 방식은 검색엔진 최적화가 어렵다.

: 포털사이트 검색엔진 크롤러가 데이터를 수집하지 못하는 경우가 많기 때문에 최근에는 구글 검색엔진에 자바스크립트 엔진이 내장되어 있어 크롤링이 된다는 이야기도 있음(구글이 공개하지 않음)

Naver 나 Daum 같은 경우 CSR 방식으로 만든 페이지의 데이터를 검색하지 못하는 이슈가 있어 별도의 보완작업이 필요.

#### SSR

- 네트워크가 느릴 때
- 검색엔진 최적화가 필요할 때
- 페이지 상호작용이 별로 없을 때

#### CSR

- 네트워크 빠를 때
- 검색엔진 최적화가 필요 없을 때
- 페이지 상호작용이 많이 필요할 때

\* React 는 CSR 방식이라 SEO 에 어려움이 있기 때문에 최근에는

NEXT.js 와 같은 프레임워크를 통해 서버사이드 렌더링을 가능하게 하는 방식을 많이 도입하고 있다.

NEXT.js 는 pre-reloading 을 통해 미리 데이터가 렌더링된 페이지를 가져올 수 있게 해주므로 사용자에게 더 좋은 경험을 주고, 검색 엔진에 잘 노출 될 수 있도록 해주는 SEO 에서도 장점을 얻을 수 있다.

## SEO (Search Engine Optimization) – 검색엔진 최적화

### 1. SEO 란 무엇인가?

- SEO(Search Engine Optimization, 검색엔진 최적화)
  - 구글, 네이버, Bing과 같은 검색엔진에서 내 웹사이트나 콘텐츠가 상위에 노출되도록 최적화하는 기법
  - 즉, "내가 만든 웹페이지를 검색엔진이 잘 이해하도록 구조화"하는 작업
- 쉽게 말해, 검색엔진에게 내 사이트를 친절하게 소개하는 기술

### 2. SEO 가 중요한 이유

- 검색엔진은 인터넷 사용자가 웹사이트를 찾는 가장 중요한 통로
- 대부분의 사용자는 검색 결과 첫 페이지(특히 상위 3~5 개 결과)만 확인한다
- SEO 를 잘하면 별도의 광고비용 없이도 **높은 방문자 유입(트래픽)** 확보 가능

### 3. SEO 의 작동 원리 (검색엔진의 기본 과정)

검색엔진은 다음 과정을 거쳐 웹페이지를 노출한다:

### 1. 크롤링 (Crawling)

- 검색엔진 로봇(크롤러, 스파이더)이 웹을 돌아다니며 웹페이지를 수집

### 2. 인덱싱 (Indexing)

- 수집한 웹페이지의 내용을 분석하고 데이터베이스에 저장

### 3. 랭킹 (Ranking)

- 사용자가 검색할 때, 인덱스된 페이지 중 **관련성·신뢰도**를 평가해 순위를 매김
- 

SEO 는 이과정에서 **검색엔진이 내 콘텐츠를 더 잘 크롤링하고, 인덱싱하고, 높은 랭킹을 부여하도록 최적화하는 활동**

## 웹 프로그래밍 시작을 위한 준비

### 1. 웹 브라우저 준비

- 웹 프로그래밍의 결과물(HTML, CSS, JavaScript)은 **웹 브라우저에서 실행**된다.
- 대표적인 브라우저 : Chrome, Edge, Firefox, Safari, Opera 등
- **크롬(Chrome)** 은 개발자 도구(DevTools)가 강력하여 가장 많이 사용된다.

### 2. 개발 도구 선택 (코딩 환경)

- 가장 단순한 방법 : **메모장**으로 작성 가능
- 하지만 효율적인 개발을 위해 다양한 **IDE / 에디터** 사용
  - Visual Studio Code
  - Eclipse
  - IntelliJ IDEA
  - Aptana Studio 등
- 우리 수업에서는 Front 는 VS Code, Back 은 Eclipse 를 사용한다.

### 3. 코드 실행 방식

#### (1) Front-End 코드 실행

- HTML, CSS, JavaScript 는 **브라우저에서 직접 실행** 가능하다.
- 예: index.html 파일을 더블 클릭 → 기본 브라우저에서 실행됨
- 이 경우는 단순히 **로컬 PC 에서 실행되는 동작**으로, 실제 웹서비스와는 다르다.

## (2) Web Server 환경 필요성

- 실제 웹서비스는 **HTTP 프로토콜 기반 Web Server** 위에서 동작해야 한다.
- 따라서 단순 파일 실행이 아니라 → **웹서버 주소 형식**으로 접근해야 한다.
  - 예: `http://localhost:8080/index.html`
- 대표적인 웹서버: **Apache, Nginx, Tomcat**

## 4. Apache Tomcat 설치와 실행

### (1) Tomcat 다운로드

- 공식 사이트: <https://tomcat.apache.org/>
- 버전: Tomcat 10 사용

### (2) 설치 방식

- **Install Version** : 운영 환경에 적합 (자동 설치)
- **Zip/압축 Version** : 개발 환경에서 적합 (간단 압축 해제 후 사용)
- 📁 우리는 압축 버전을 사용한다.
- 

### (3) 실행 포트 주의

- Tomcat 기본 포트 번호: **8080**
- 단, **Oracle DB** 등 다른 서비스가 이미 **8080** 을 사용 중일 수 있음
- 충돌 방지 → Tomcat 설정 파일 `server.xml` 에서 포트 번호 변경 필요

### 정리하면

1. 브라우저 준비 → 개발 결과 확인
2. 개발 도구 선택 → VS Code, Eclipse 사용
3. 코드 실행 → 단순 실행은 로컬 동작, 실제 서비스는 웹서버 필요
4. Tomcat 설치 및 설정 → 웹 애플리케이션 실행 환경 구축
- 5.

이 과정을 통해 "파일 실행 수준"이 아닌, 실제 **웹 서비스 환경**에서 웹 프로그래밍을 시작할 수 있다.

## VS Code 에서 Live Server 사용하기

### 1. 왜 Live Server 가 필요할까?

#### (1) 기본 실행 방식의 한계

- VS Code 에서 작성한 `index.html` 파일을 **더블 클릭**하면 브라우저에서 실행되긴 한다.

- 하지만 이 경우는 단순히 **로컬 파일(local file://)** 로 열리는 것이지, **웹서버 환경(http://)** 에서 실행되는 것이 아니다.

### 문제점

- **상대 경로 참조 문제** : CSS, JS 파일을 불러올 때 에러 발생 가능
- **Ajax 요청 제한** : 로컬 파일 실행 시 CORS 정책 때문에 Ajax 동작이 제한될 수 있음
- **자동 새로고침 불가** : 파일 수정 후 브라우저를 직접 새로고침해야 반영됨

### (2) Live Server 의 필요성

- Live Server 는 **로컬 개발용 웹 서버**를 실행해준다.
- 파일을 http://localhost:5500 형태로 실행할 수 있다.
- 코드 저장 시 브라우저가 자동으로 새로고침(Hot Reload) 된다.
- 

즉, 실제 웹 서버 환경과 유사하게 개발할 수 있도록 도와주는 도구이다.

## 2. Live Server 설치 방법

1. VS Code 실행
2. 왼쪽 **Extensions(확장 프로그램)** 아이콘 클릭
3. 검색창에 "**Live Server**" 입력
4. **Ritwick Dey** 가 만든 "Live Server" 선택 후 **Install(설치)**

## 3. Live Server 실행 방법

1. 실행할 HTML 파일 열기 (index.html)
2. 마우스 오른쪽 클릭 → "**Open with Live Server**" 선택
3. 브라우저가 자동 실행되며, 주소는 보통
4. http://127.0.0.1:5500/index.html
5. 또는
6. http://localhost:5500/index.html
7. 코드 수정 후 저장하면 → 브라우저가 **자동 새로고침** 되어 변경 사항 반영

## 4. Live Server 장점 정리

- 실제 웹 서버 환경(http://)에서 실행 가능
- 자동 새로고침 지원 → 개발 생산성 ↑
- Ajax / Fetch / API 요청 테스트 가능
- 멀티 브라우저 테스트 가능

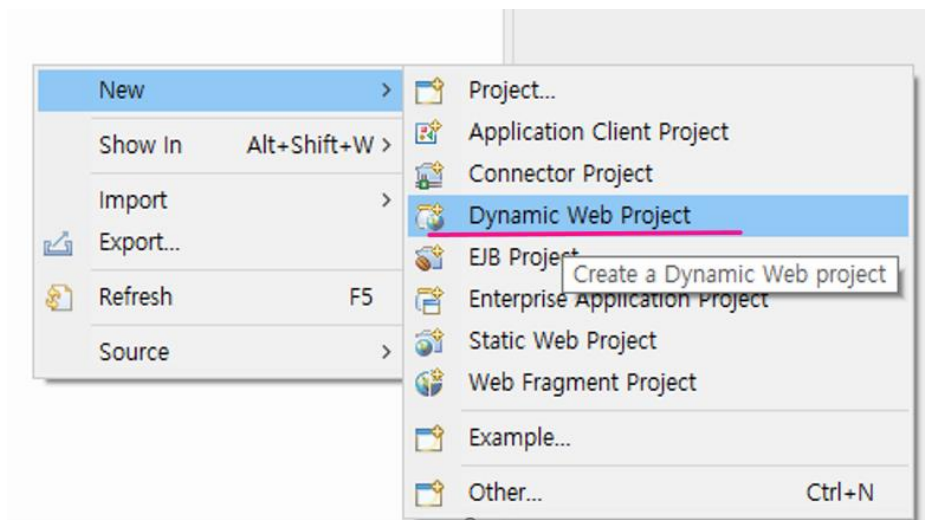
## 결론

VS Code 자체만으로는 단순히 파일을 여는 수준이지만,

**Live Server** 를 활용하면 웹 서버 환경을 시뮬레이션할 수 있다.

따라서 웹 프로그래밍 학습 및 프로젝트 개발 시 반드시 필요한 필수 확장 도구이다.

## eclipse 에서 프로젝트 생성



## 이클립스 Dynamic Web Project 디렉터리 구조

Dynamic Web Project 는 **프론트엔드(HTML, CSS, JS, JSP)** 와 **백엔드(Servlet, Java Class, Library)** 를 함께 포함하는 구조로 만들어진다.

이 구조는 **Maven 표준 디렉터리 구조**를 기반으로 하며, 실제 웹 애플리케이션이 **Tomcat(WAS)** 위에서 동작할 수 있도록 설계된다.

### 1. src/main/java

- **Servlet 클래스와 Java 관련 소스 코드(.java) 파일을 저장하는 디렉터리**
- 예:
  - Controller(Servlet)
  - DAO, DTO/VO, Service 클래스 등
- 컴파일 후 .class 파일은 WEB-INF/classes 폴더에 배치된다.

### 2. src/main/webapp

- 웹 애플리케이션의 Root(루트) 디렉터리
- 브라우저가 직접 접근할 수 있는 프론트엔드 리소스들이 위치
  - HTML
  - CSS
  - JavaScript
  - JSP 문서

즉, 사용자가 `http://localhost:8080/프로젝트명/파일명` 으로 접근했을 때 가장 먼저 참조되는 위치이다.

### 3. src/main/webapp/WEB-INF

- 브라우저가 직접 접근할 수 없는 영역
- 외부 사용자에게 노출되면 안 되는 중요한 자원들을 저장
- 특징:
  - 이곳에 HTML, JSP, CSS 등을 넣으면 브라우저에서 직접 실행 불가
  - 보안 목적: 클라이언트가 직접 요청할 수 없고, 반드시 Servlet 을 통해서만 접근 가능

### 4. src/main/webapp/WEB-INF/lib

- 백엔드에서 사용하는 라이브러리(.jar) 파일 저장소
- 예:
  - 데이터베이스 연동용 라이브러리 (ex. ojdbc8.jar)
  - JSON 처리 라이브러리
  - 기타 외부 API 라이브러리

Tomcat 실행 시 이 lib 안의 JAR 파일이 클래스패스(Classpath)에 포함되어 실행된다.

## 정리하면

디렉터리	역할
src/main/java	Servlet 및 자바 클래스 소스 코드
src/main/webapp	웹의 Root, HTML/CSS/JS/JSP 저장
WEB-INF	보안 영역, 브라우저 직접 접근 불가
WEB-INF/lib	프로젝트에서 사용하는 백엔드 라이브러리(JAR) 저장

## 핵심 포인트

- src/main/webapp = 웹 Root
- WEB-INF = 브라우저 직접 접근 불가 → 보안 디렉터리
- WEB-INF/lib = 백엔드 라이브러리 저장소

## 알아두면 좋은 웹 개발 사이트

### 1. [W3Schools](#)

웹 기술을 온라인으로 학습할 수 있는 대표적인 교육용 웹사이트.

### 2. [MDN Web Docs](#)

모질라 재단과 여러 IT 기업이 함께 만든 웹 개발 공식 문서 저장소로, 프로그래밍 학습에 필수적인 자료 제공.

### 3. [Stack Overflow](#)

전 세계 개발자들이 질문과 답변을 공유하는 커뮤니티. 프로그래밍 중 막힐 때 해결책을 찾을 수 있는 대표 사이트.

### 4. [JS Bin](#)

HTML, CSS, JavaScript 코드를 온라인에서 작성하고 즉시 실행·테스트할 수 있는 웹서비스.

### 5. [JSON Placeholder](#)

연습용 샘플 데이터를 제공하는 무료 REST API 서비스. Ajax, Fetch, Axios 테스트에 유용.

### 6. [Picsum Photos](#)

무료 샘플 이미지를 간단히 가져올 수 있는 서비스. 디자인 시안이나 프론트엔드 연습에 활용.

### 7. [Can I Use](#)

HTML, CSS, JavaScript 기능이 각 브라우저에서 지원되는지 확인할 수 있는 사이트. 크로스 브라우저 이슈 점검에 필수.

### 8. [Image Color Picker](#)

이미지에서 원하는 색상을 추출해 색상 코드(HEX, RGB 등)를 알려주는 도구.