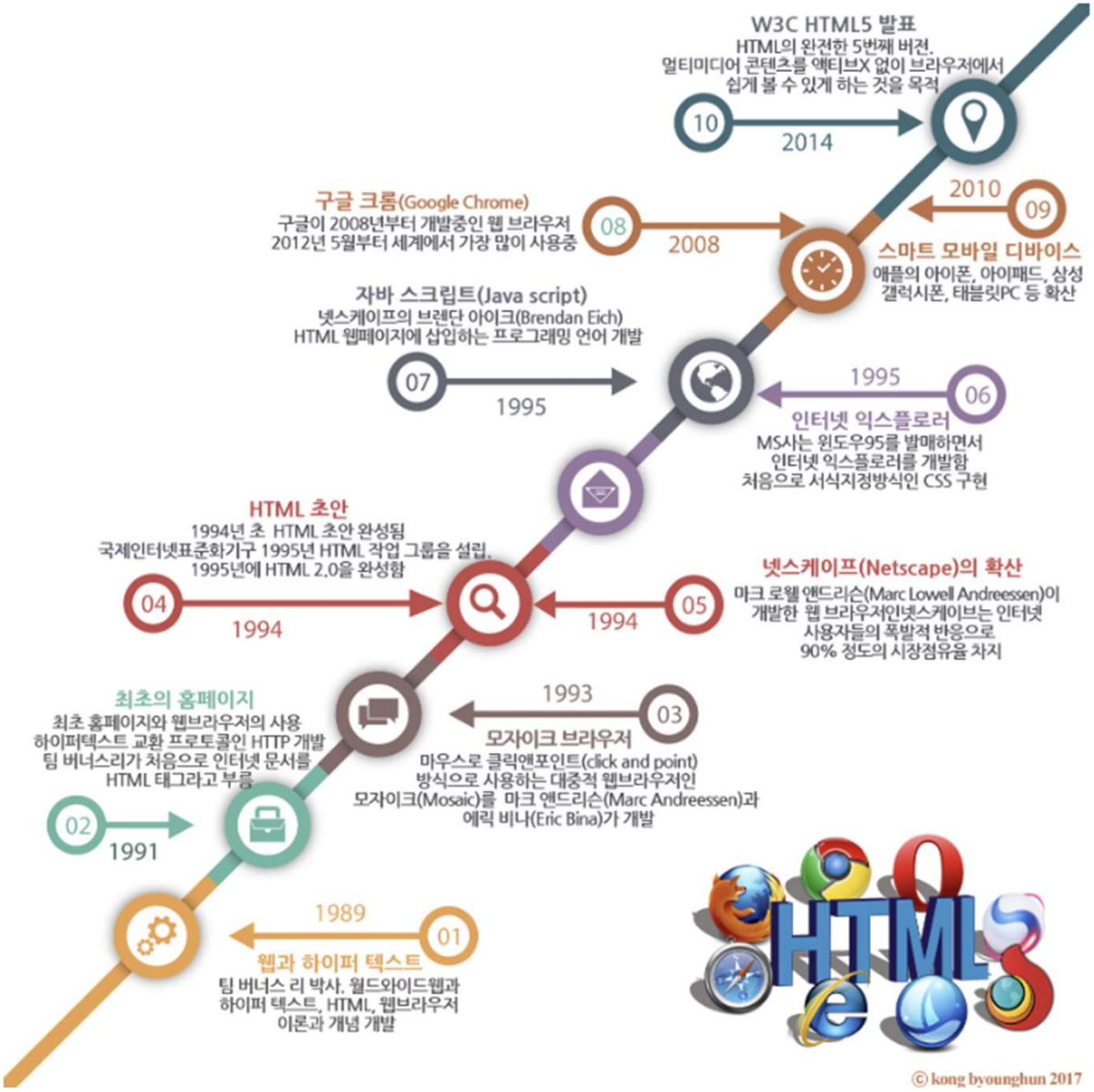
01

JavaScript 이해

목차

Javascript 이해

- 인터넷과 HTML의 역사



Javascript 이해

● 인터넷과 HTML의 역사

구분	특징
초기 웹 (Web 1.0) 1990년대 초 - 중반	월드 와이드 웹(WWW)의 탄생 HTML 페이지는 간단한 텍스트와 하이퍼링크로 구성, 정적 웹 페이지
Web 2.0 2000년대 초반	동적 콘텐츠 및 사용자 참여 블로그, 소셜 미디어, 위키와 같은 플랫폼이 등장 AJAX 풍부한 경험을 제공하는 Rich Internet Application(Flash, Sliver Light, Java Applet 등을 이용)
웹 3.0 (세만틱 웹) 2010년대	플러그인의 기능이 HTML5 자바스크립트 API로 구현가능해졌고, AJAX의 활성화로 SPA(Single Page Application)로 웹 어플리케이션이 개발이 활성화됨 Node.js로 서버 단에서도 런타임 환경에서 자바스크립트 동작이 가능해짐 데이터를 더 잘 이해하고 처리할 수 있는 지능형 시스템을 목표로 합니다 시맨틱 웹 : 데이터의 의미를 이해하고 자동으로 처리할 수 있게 하는 기술 사용자의 취향과 행동에 맞춘 개인화된 콘텐츠와 서비스가 제공

Javascript 이해

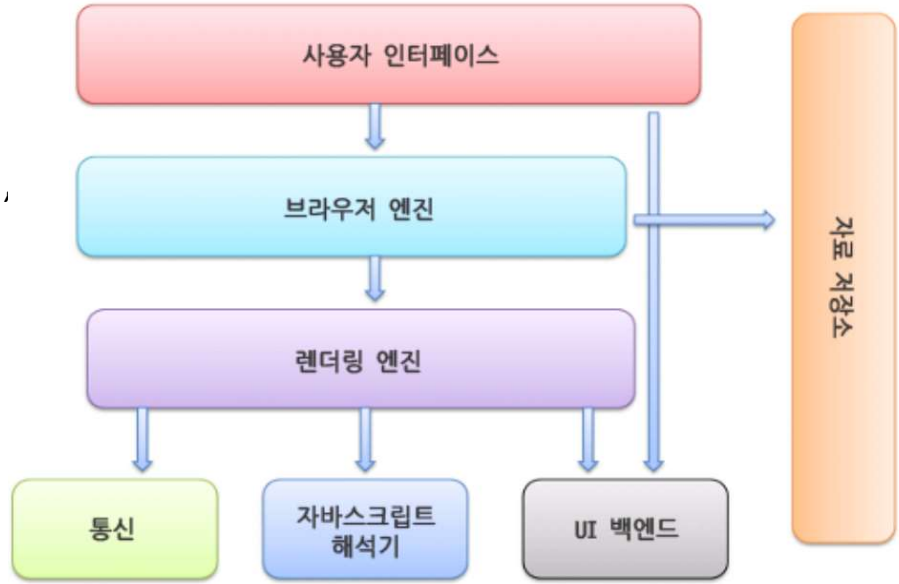
● 인터넷과 HTML의 역사

구분	특징
웹 4.0 (미래 웹)	다양한 기기와 센서들이 인터넷에 연결되어, 데이터가 상호 교환되고 처리됩니다 웹 환경에서 VR과 AR 기술을 통해 새로운 형태의 사용자 경험이 제공됩니다. 웹이 사용자의 요구를 예측하고, 더 복잡한 작업을 자동으로 수행할 수 있게 하는 발전된 AI 기술이 적용됩니다. 블록체인과 같은 기술을 통해 더 안전하고 분산된 형태의 웹이 구현됩니다.

Javascript 이해

Browser 구조

- 사용자 인터페이스(user Interface) : 검색창, 뒤로가기/앞으로 가기 버튼, 새로 고침 등 브라우저 프로그램 자체의 GUI를 구성하는 부분
- 브라우저 엔진(Browser Engine) - 사용자 인터페이스와 렌더링 엔진 사이의 동작을 제어
- 렌더링 엔진(Rndering Engine) - 요청한 콘텐츠(HTML/CSS 등)를 파싱(parsing)하여 표시한다.
- 통신(Networking) - HTTP 요청과 같은 네트워크 호출에 사용된다.
- JS 엔진(JS Engine) - 자바스크립트 코드를 해석하고 실행한다.
- UI 백엔드(UI Backend) - 기본적인 위젯을 그리는 인터페이스로 플랫폼에서 명시하지 않은 일반적인 인터페이스
- 데이터 저장소(Data Storage) : 로컬 스토리지, 쿠키 등 브라우저 메모리를 활용하여 저장하는 영역

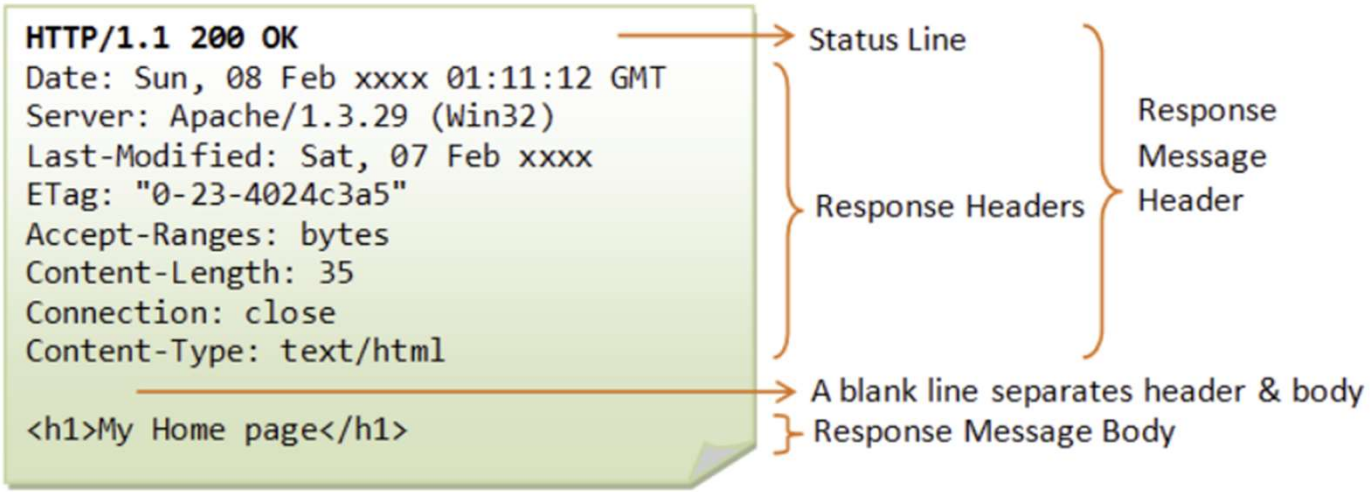


브라우저의 주요 구성 요소

Javascript 이해

HTTP Request/Response Message구조

- Request Line(요청 라인) : 데이터 처리 방식(HTTP Method)와 기본 페이지, 프로토콜 버전이 포함된다.
- Request Header(요청 헤더) : User-Agent, Accept, Cookie, Referer, Host 정보가 포함된다
- A blank line separates header & body(공백 라인) : 헤더와 본문을 구분하기 위한 캐리지 리턴(₩r)과 라인 피드(₩n)으로 구성
- Request Message Body(메세지 본문) : 요청과 관련된 정보가 담겨있다



Javascript 이해

HTTP Request/Response Message구조

- start line : HTTP Request Message의 시작 라인

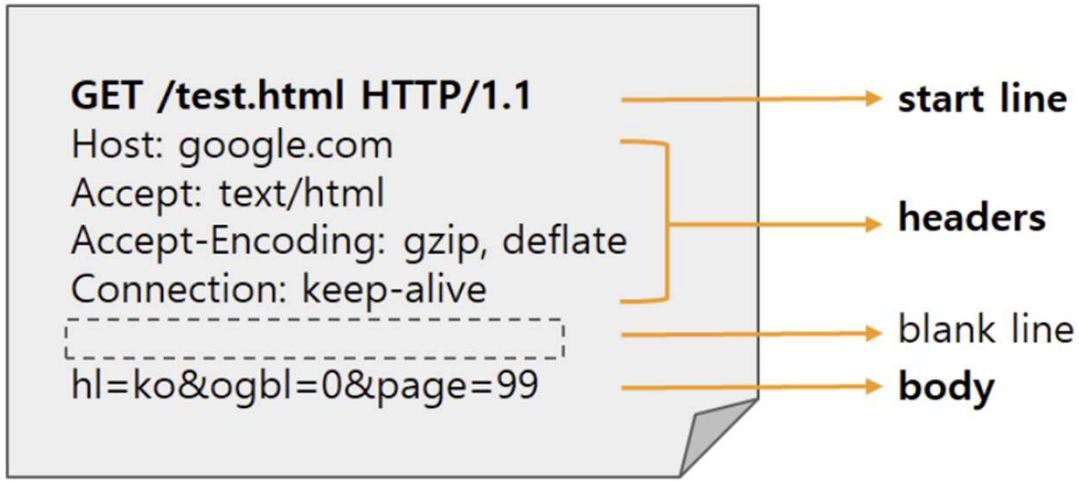
  HTTP method, Request target, HTTP version 3가지 부분으로 구성

- Headers : 해당 request에 대한 추가 정보(addtional information)를 담고 있는 부분

- Body : HTTP Request가 전송하는 데이터를 담고 있는 부분

  전송하는 데이터가 없다면 body 부분은 비어있습니다.

  보통 post 요청일 경우, HTML 폼 데이터가 포함되어 있습니다.



Javascript 이해

- **JavaScript**
  - 모질라 재단에서 개발한 프로토타입 기반의 스크립트 언어
  - 동적 타입의 인터프리터 언어로 런타임에서 오류를 발견

Javascript 이해

● JavaScript 발전

구분	특징
JavaScript의 탄생 1995년	넷스케이프 사의 브랜든 아이히Brendan Eich에 의해 모카라는 이름으로 웹 페이지에 동적인 요소를 추가하고 사용자 상호작용을 처리하기 위한 간단한 스크립팅 언어로 개발 (LiveScript로 이름변경) SunMicroSystem와 공동으로 라이브스크립트를 확장한 JavaScript 탄생
1997년 - 2005년 표준화 및 초기 발전	ECMAScript 표준화 : ECMAScript 1(1997년), ECMAScript 2(1998년), ECMAScript 3(1999년) 발표
AJAX와 웹 2.0 2005년 - 2010년	동적인 웹 애플리케이션의 발전 jQuery, Prototype, MooTools 등의 JavaScript 라이브러리가 등장 더 쉽게 크로스 브라우저 코드를 작성
2009년 - 2015년 ECMAScript 5 (ES5) Node.js	ES5는 JSON, Object.defineProperty, Array 메서드 등을 도입하여 언어의 기능을 대폭 강화 2009년, Ryan Dahl이 발표, JavaScript를 서버 측에서 사용
ECMAScript 6 (ES6/ES2015)	let/const, 화살표 함수, 클래스, 모듈, Promise, 템플릿 리터럴 기능을 도입 Webpack, Browserify 등의 도구가 등장하여, 모듈화된 JavaScript 코드를 쉽게 번들링하고 관리 Angular, React, Vue.js와 같은 프레임워크와 라이브러리의 발전

Javascript 이해

JavaScript 발전

- AWS Lambda, Google Cloud Functions 등의 서버리스 플랫폼에서 JavaScript를 활용하여 백엔드 로직을 구현하는 것이 보편화되고 있습니다.
- TensorFlow.js와 같은 라이브러리를 통해, 브라우저에서 직접 머신러닝 모델을 실행하고, AI 기반의 웹 애플리케이션을 개발하는 사례가 늘어나고 있습니다.
- JavaScript 외에도 다른 언어로 작성된 코드를 웹에서 실행할 수 있는 WebAssembly(Wasm)가 등장하며, JavaScript와의 상호작용이 활발히 이루어지고 있습니다.
- JavaScript는 웹, 서버, 데스크톱, 모바일 등 다양한 플랫폼에서 사용되는 범용 프로그래밍 언어로 발전

Javascript 이해

- **Browser에서 실행되는 JavaScript**
  - 클라이언트쪽에서 독립적으로 실행되는 웹 문서(XHTML, HTML)에 끼워서 사용하는 스크립트 언어
  - 플랫폼에 독립적
  - 브라우저 호환성(Browser Compatibility) – 모든 웹 브라우저에서 동일한 실행결과
  - 인터프리터 언어의 형태
  - 변수의 타입이 런타임에 결정되는 동적 타이핑(Dynamic Typing)
  - 함수가 일급 객체로 취급하는 함수형 프로그래밍(Functional Programming)
  - 클래스 문법을 이용해서 객체를 정의, 클래스를 상속할 수 있다
  - 콜백 함수, 프로미스, async/await를 통해 비동기 작업 처리
  - DOM 조작 - HTML 문서의 구조를 동적으로 변경 가능
  - React, Angular, Vue.js 등 다양한 프레임워크와 라이브러리 제공

Javascript 이해

● 자바 스크립트 활용

- HTML 및 CSS와 함께 사용되어 웹 페이지를 동적으로 만듭니다.
- 사용자 인터페이스와 백엔드 서버 간의 통신을 가능하게 하고, 브라우저에서 실행되는 애플리케이션을 만들 수 있습니다.
- React Native, Ionic 및 PhoneGap 프레임워크 및 플랫폼을 이용하여 모바일 애플리케이션을 개발할 수 있습니다.
- D3.js와 같은 도구를 사용하여 데이터를 시각적으로 표현하는 데 사용될 수 있습니다
- HTML5 Canvas와 WebGL과 같은 기술을 사용하여 그래픽과 애니메이션을 생성하고 게임 로직을 구현할 수 있습니다.
- 독립적 application
- 딥러닝 라이브러리

Javascript 이해

Web Client Side의 자바스크립트의 기능

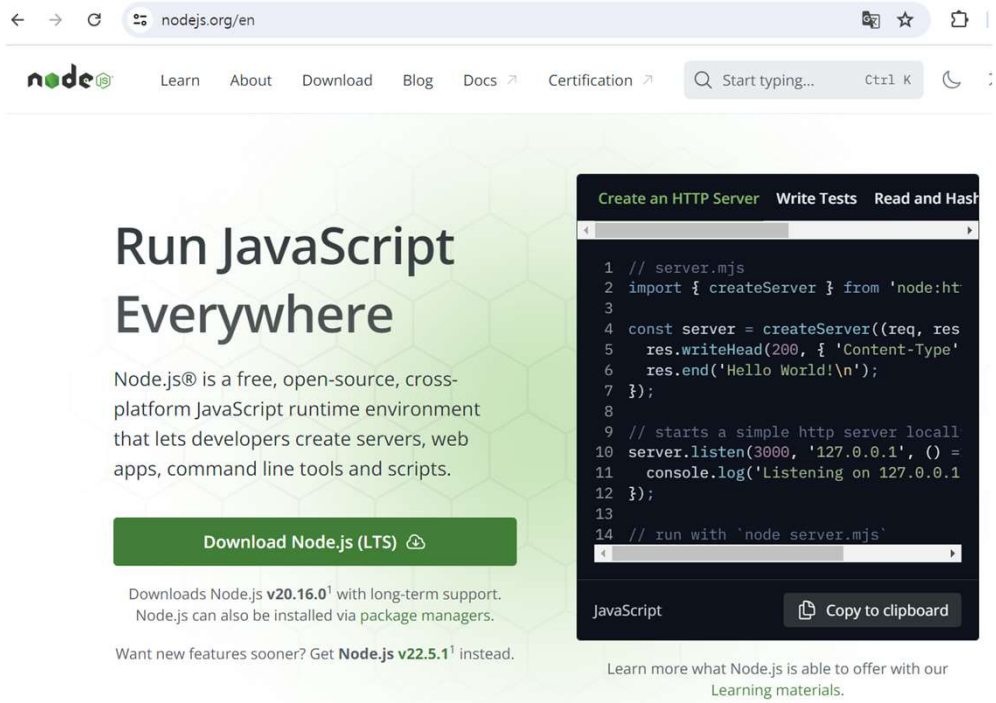
- 웹 문서에 동적으로 움직일 수 있는 프로그램적인 요소를 추가
- 어떤 이벤트에 대한 반응을 할 수 있도록 웹 문서를 구성
- 웹 문서 내의 HTML 요소의 내용을 읽거나 변경
- 클라이언트의 정보를 서버에 보내기 전에 먼저 데이터 검증
- 클라이언트가 어떤 웹 브라우저를 사용하는지 알아내어 사용자의 디바이스 웹 브라우저에 맞는 웹 문서를 호출하는 맞춤형 서비스를 제공 (반응형 웹 컨텐츠)
- 클라이언트 컴퓨터에 쿠키를 새로 만들거나 생성된 쿠기를 삭제 및 회수할 수 있다.

Javascript 이해

- **HTML5 JavaScript API**
  - Geolocation API : 사용자 위치 정보를 제공
  - Web Storage API : 클라이언트 측 데이터 저장
  - Canvas API : 게임, 데이터 시각화, 그래픽 편집 등 다양한 그래픽 작업 가능
  - WebSockets API : 서버와의 양방향 통신을 가능하게 함
  - File API : 파일 시스템에 접근하여 파일 읽기/쓰기 기능 제공
  - IndexedDB API : 클라이언트 측 대용량 데이터베이스 관리 기능 제공
  - Drag and Drop API : HTML 요소의 드래그 앤 드롭 기능
  - History API : 브라우저의 세션 기록을 조작, 페이지 리로드 없이 URL 변경
  - Web Workers API : 백그라운드 스레드에서 자바스크립트 코드 실행

Javascript 라이브러리

- **Node.js**
  - 2008년 9월 : 구글은 크롬 웹 브라우저의 베타 버전을 발표
  - 2009년 1월 이후 : 자바스크립트를 웹 브라우저가 아닌 곳에서도 사용할 수 있는 표준안 제시
  - CommonJS 표준 발표 이후 : CommonJS 표준과 V8 자바스크립트 엔진을 기반으로 Node.js를 개발



Javascript 라이브러리

Node.js 설치

- https://nodejs.org/
- REPL(Read-Eval-Print Loop) : Node.js 코드를 즉시 실행하고 결과를 확인할 수 있는 환경을 제공하는 대화형 콘솔로 빠른 테스트와 디버깅에 유용

REPL 명령어	설명
.help	REPL에서 사용할 수 있는 모든 명령어 목록을 표시합니다
.editor	여러 줄의 코드를 입력할 수 있는 편집 모드로 전환합니다. 편집 모드를 종료하려면 Ctrl+D를 누릅니다.
.break	현재 입력 줄을 중단하고 새로운 입력을 시작합니다.
.clear	현재 REPL 세션에서 모든 변수를 지우고 컨텍스트를 초기화합니다.
.exit	REPL을 종료합니다. Ctrl+C를 두 번 눌러서도 종료할 수 있습니다.
.load	외부 파일을 로드하고 실행합니다.
.save	현재 REPL 세션을 파일로 저장합니다.

Javascript 라이브러리

Node.js

- V8 자바스크립트 엔진(구글 크롬에서 사용하는 엔진) 위에서 작동하는 자바스크립트 런타임
- 비동기식, 논블로킹 I/O 모델을 사용하여 고성능, 고확장성 서버를 구현 가능
- 싱글 스레드 이벤트 루프를 사용
- 크로스 플랫폼 지원
- NPM(Node Package Manager) 제공 - 오픈 소스 라이브러리와 패키지를 쉽게 설치하고 관리
- CommonJS 모듈 시스템을 사용하여 코드를 모듈화하고 재사용할 수 있다.
- ES6 모듈 시스템도 지원
- HTTP 서버, RESTful API 서버, GraphQL 서버 등 구현에 사용됨
- 채팅 애플리케이션, 실시간 협업 도구, 라이브 스트리밍 서비스 구현에 사용됨
- 각각의 서비스가 독립적으로 배포되고 확장 가능한 마이크로서비스를 구축하는 데 적합
- 명령줄 인터페이스 도구를 만들기 위한 환경으로도 사용됨

Javascript 라이브러리

TypeScript

- 마이크로소프트에서 구현한 자바스크립트 슈퍼셋 프로그래밍 언어
- 자바스크립트 문법에 타입스크립트 문법을 추가한 언어
- 정적 타입의 컴파일 언어로 타입스크립트 컴파일러 또는 바벨(Babel)을 통해 자바스크립트 코드로 컴파일합니다
- 변수나 함수에 타입을 명시하여 컴파일 시점에 오류를 발견하여 자바스크립트 개발의 생산성을 높일 수 있습니다.
- Visual Studio Code의 경우 툴의 내부가 타입스크립트로 작성되어 있기 때문에 타입스크립트 개발에 최적화 되어 있습니다.

```
function sum(a: number, b: number) {
    return a + b;
}

sum('1', '2'); //error
```

Javascript 라이브러리

TypeScript

- 타입스크립트 개발 환경을 구축하려면 Node.js가 먼저 설치되어 있어야 합니다.
- 타입스크립트의 확장자는 .ts입니다
- tsc 명령어로 컴파일 합니다.
- 컴파일이 완료되면 자바스크립트 파일(.js)이 생성됩니다.
- tsconfig.json 설정파일에 컴파일 옵션을 설정하여 컴파일 할 수 있습니다.
- tsc --init 명령어로 default 설정파일을 생성할 수 있습니다.
- https://www.typescriptlang.org/play

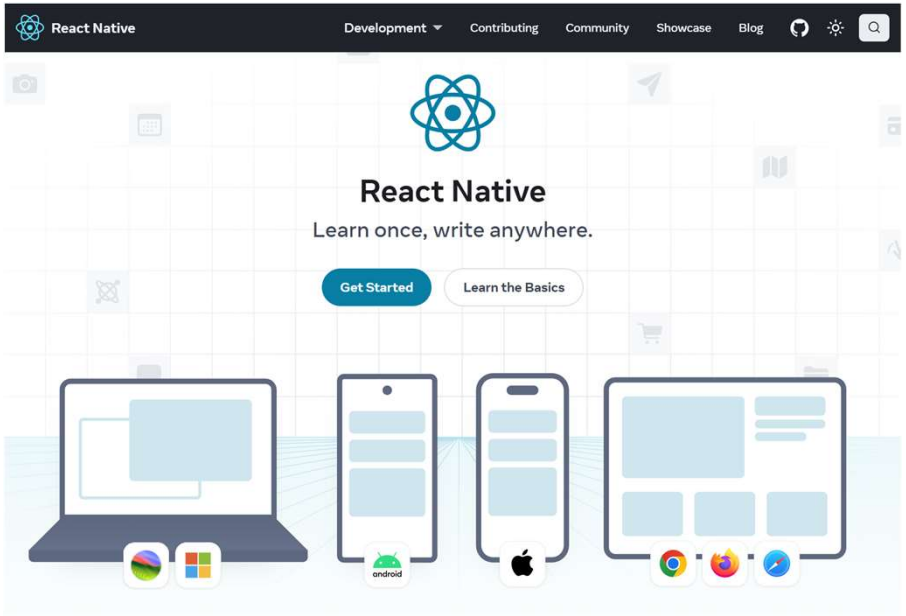
```
$ npm install -g typescript@4.9.5
```

- 컴파일 과정 없이 바로 실행하는 ts-node도 내부적으로는 타입스크립트를 자바스크립트로 변환 후 실행합니다.

Javascript 로 할 수 있는 일

● 모바일 애플리케이션 개발

- React Native는 Facebook에서 개발하고 유지보수 중.
- 자바스크립트를 사용하여 iOS와 Android 애플리케이션을 동시에 개발.
- 하나의 코드베이스로 두 플랫폼에서 네이티브와 같은 성능 제공.
- 큰 커뮤니티와 풍부한 라이브러리 생태계.
- JSX 문법을 사용한 컴포넌트 기반 개발.
- Hot Reloading 기능을 통해 빠른 개발 사이클 제공.
- Ionic, Flutter (with Dart), NativeScript, Cordova (PhoneGap), Quasar Framework, Xamarin (with C#), Framework7



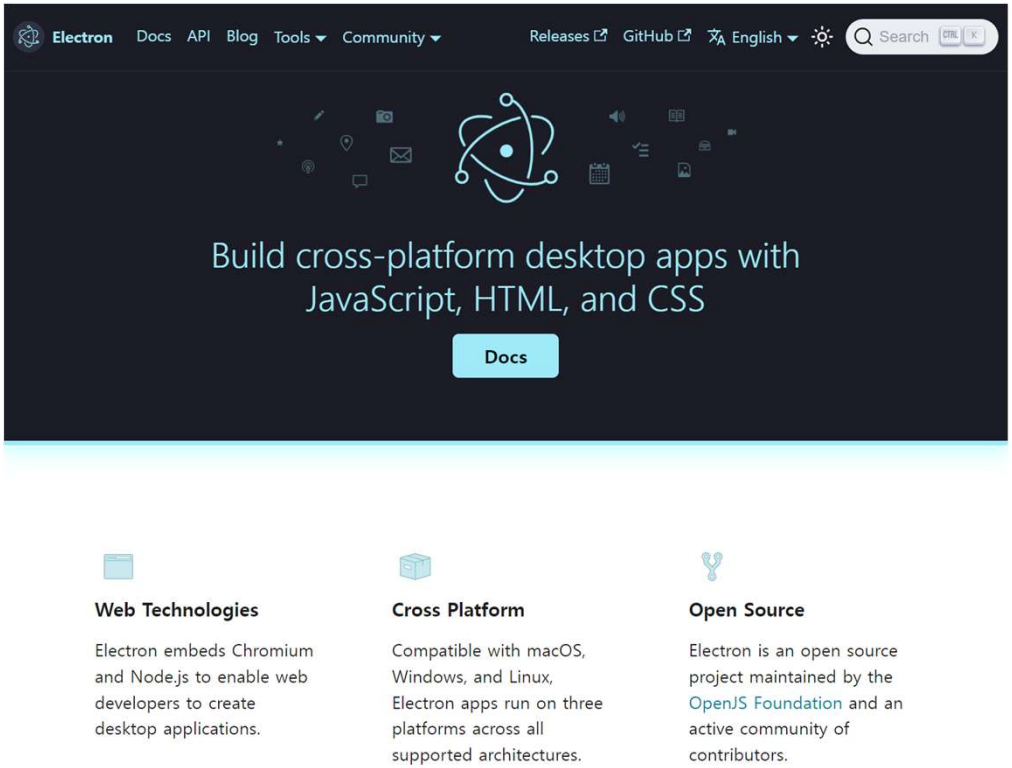
Javascript 로 할 수 있는 일

● 모바일 애플리케이션 개발

- React Native는 Facebook에서 개발하고 유지보수 중.
- 자바스크립트를 사용하여 iOS와 Android 애플리케이션을 동시에 개발.
- 하나의 코드베이스로 두 플랫폼에서 네이티브와 같은 성능 제공.
- 큰 커뮤니티와 풍부한 라이브러리 생태계.
- JSX 문법을 사용한 컴포넌트 기반 개발.
- Hot Reloading 기능을 통해 빠른 개발 사이클 제공.

Javascript 로 할 수 있는 일

- **데스크톱 애플리케이션 개발**
  - Electron : 웹 기술(HTML, CSS, JavaScript)을 사용하여 크로스 플랫폼 데스크탑 애플리케이션을 개발할 수 있게 해주는 프레임워크
  - Chromium 브라우저 엔진을 내장하여 최신 웹 표준을 지원
  - 크로스 플랫폼
  - Node.js 통합
  - Visual Studio Code, Slack, Trello, Discord, Atom 에 사용



Javascript 로 할 수 있는 일

● 게임 개발 라이브러리와 프레임워크

- Phaser : 2D 게임 개발에 최적화된 프레임워크
- Three.js : 3D 그래픽과 애니메이션을 위한 라이브러리
- Babylon.js : 3D 게임과 그래픽을 위한 강력한 엔진
- PixiJS : 빠르고 유연한 2D 렌더링 라이브러리
- PlayCanvas : 클라우드 기반의 3D 게임 엔진 및 개발 플랫폼으로 VR 및 AR 지원
- MelonJS : 경량의 HTML5 기반 2D 게임 엔진
- Construct (with JavaScript) : 비주얼 스크립팅을 지원하는 2D 게임 개발 도구
- A-Frame : VR/AR 콘텐츠 개발을 위한 프레임워크

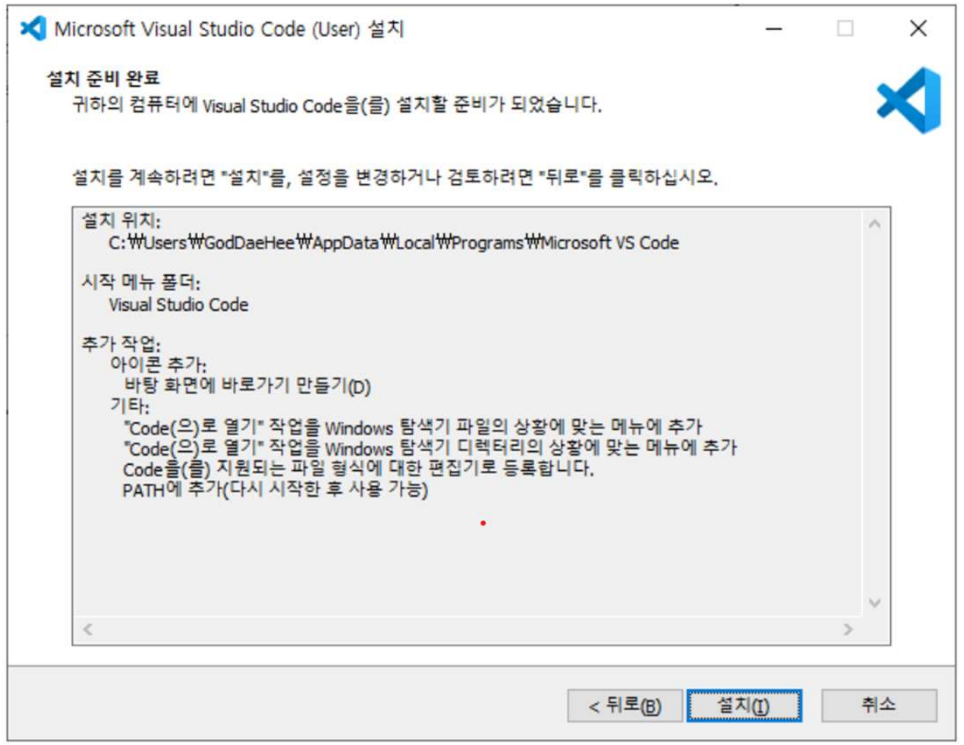
유니티가 지원하는 플랫폼

실습 환경 구축

실습 환경

● 코드 편집기

- Visual Studio Code는 Microsoft에서 개발한 무료 오픈소스 코드 편집기
- 높은 확장성과 개인화 가능성
- 다양한 프로그래밍 언어 지원
- 코드 작성 및 디버깅 기능
- Git과 같은 버전 관리 시스템 연동 기능
- 강력한 IntelliSense 기능
- https://code.visualstudio.com/download



실습 환경

● 확장(extension)

- 단축키 : Ctrl+Shift+X
- ESLint : 코드 품질 검사 및 스타일 일관성을 유지하는 데 도움을 주는 Extension
- Prettier : Code formatter, 코드를 일관된 스타일로 포맷팅
- Live Server : HTML 파일을 변경할 때마다 자동으로 브라우저를 새로 고침하여 변경 사항을 실시간으로 확인할 수 있습니다.
- Debugger for Chrome : VSCode에서 Chrome 브라우저와 연결하여 디버깅을 쉽게 할 수 있도록 도와주는 확장
- racket Pair Colorizer 2 : 중괄호, 대괄호 등의 쌍을 색깔별로 표시하여 코드 가독성을 높입니다.
- JavaScript (ES6) code snippets
- 한글 팩

Web Project 구조

● 장고(Django) 프로젝트의 기본 구조

- 프로젝트 루트 디렉토리

- manage.py : 장고 프로젝트의 관리를 위한 커맨드 라인 유틸리티
  서버를 실행하거나 데이터베이스 마이그레이션을 관리하는 등의 작업을 수행할 수 있습니다.

- 프로젝트 설정 디렉토리

- __init__.py : 파이썬 패키지로 인식되도록 합니다.

- settings.py : 프로젝트의 설정 파일
  데이터베이스 설정, 설치된 앱 목록, 미들웨어, 템플릿 설정 등 다양한 설정을 포함합니다.

- urls.py: URL 라우팅 설정 파일입니다. URL과 뷰를 매핑하는 역할을 합니다.

- asgi.py: ASGI(Asynchronous Server Gateway Interface) 설정 파일입니다.
  비동기 서버와 통신하는 설정을 포함합니다.

```
<프로젝트명>/
    manage.py
    <프로젝트명>/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
    app_name/
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
        migrations/
            __init__.py
```

Web Project 구조

● 장고(Django) 프로젝트의 기본 구조

```
<프로젝트명>/
    manage.py
    <프로젝트명>/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
    app_name/
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
        migrations/
            __init__.py
```

- wsgi.py: WSGI(Web Server Gateway Interface) 설정 파일입니다.
  장고 프로젝트를 WSGI 호환 웹 서버와 통신하도록 합니다.
- 애플리케이션 디렉토리 (<프로젝트명>/app_name/)
- __init__.py: 이 디렉토리가 파이썬 패키지로 인식되도록 합니다.
- admin.py: 관리자 인터페이스에 모델을 등록하는 파일입니다.
- apps.py: 애플리케이션 설정 파일입니다.
- models.py: 데이터베이스 모델을 정의하는 파일입니다.
- tests.py: 테스트 케이스를 작성하는 파일입니다.
- views.py: 뷰 함수를 작성하는 파일입니다.
- migrations/: 데이터베이스 마이그레이션 파일을 저장하는 디렉토리입니다
- 템플릿 디렉토리 (templates/) : HTML 템플릿 파일을 저장하는 디렉토리입니다.
- 정적 파일 디렉토리 (static/) : CSS, JavaScript, 이미지 파일 등 정적 파일을 저장하는 디렉토리입니다.
- 미디어 파일 디렉토리 (media/) : 사용자 업로드 파일을 저장하는 디렉토리입니다.

Web Project 구조

- **Node.js 서버 프로젝트의 기본 구조**
  - 루트 디렉토리 (project/)
  - node_modules/: 프로젝트에서 사용되는 모든 의존성 패키지를 저장하는 디렉토리입니다.
  - public/: 정적 파일(CSS, JavaScript, 이미지 등)을 저장하는 디렉토리입니다.
  - src/: 애플리케이션의 소스 코드를 저장하는 디렉토리입니다.
  - controllers/: 요청을 처리하고 응답을 반환하는 로직을 담은 컨트롤러 파일을 저장하는 디렉토리입니다.
  - middlewares/: 미들웨어 함수를 저장하는 디렉토리입니다.
  - models/: 데이터베이스 모델을 정의하는 파일을 저장하는 디렉토리입니다.
  - routes/: URL 경로와 컨트롤러를 매핑하는 라우터 파일을 저장하는 디렉토리입니다.
  - services/: 비즈니스 로직을 처리하는 서비스 파일을 저장하는 디렉토리입니다.

```
project/
    ├── node_modules/
    ├── public/
    │   ├── css/
    │   ├── js/
    │   └── images/
    ├── src/
    │   ├── controllers/
    │   ├── middlewares/
    │   ├── models/
    │   ├── routes/
    │   ├── services/
    │   ├── utils/
    │   ├── views/
    │   └── app.js
    ├── .env
    ├── .gitignore
    ├── package.json
    ├── package-lock.json
    └── README.md
```

Web Project 구조

- **Node.js 서버 프로젝트의 기본 구조**
  - utils/: 유틸리티 함수나 헬퍼 함수 등을 저장하는 디렉토리입니다.
  - views/: 템플릿 파일을 저장하는 디렉토리입니다. (예: Pug, EJS 등)
  - app.js: 애플리케이션의 진입점 파일입니다. Express 애플리케이션을 설정하고 서버를 시작하는 로직이 포함됩니다.
  - app.js: 애플리케이션의 진입점 파일입니다. Express 애플리케이션을 설정하고 서버를 시작하는 로직이 포함됩니다.
  - .env: 환경 변수 파일입니다. 데이터베이스 연결 정보, API 키 등 민감한 정보를 저장합니다.
  - .gitignore: Git이 추적하지 않을 파일과 디렉토리를 지정하는 파일입니다.
  - package.json: 프로젝트의 메타데이터와 의존성 정보를 담고 있는 파일입니다.
  - package-lock.json: 의존성 트리의 정확한 버전을 기록하는 파일입니다.
  - README.md: 프로젝트에 대한 설명, 설치 방법, 사용 방법 등을 담고 있는 문서 파일입니다.

```
project/
    ├── node_modules/
    ├── public/
    │   ├── css/
    │   ├── js/
    │   └── images/
    ├── src/
    │   ├── controllers/
    │   ├── middlewares/
    │   ├── models/
    │   ├── routes/
    │   ├── services/
    │   ├── utils/
    │   ├── views/
    │   └── app.js
    ├── .env
    ├── .gitignore
    ├── package.json
    ├── package-lock.json
    └── README.md
```

Web Project 구조

- **Java Maven 프로젝트의 기본 구조**
  - 루트 디렉토리 (project/)
  - src/: 소스 코드와 리소스 파일을 저장하는 디렉토리입니다.
  - main/: 메인 애플리케이션 코드를 저장하는 디렉토리입니다.
  - java/: Java 소스 파일을 저장하는 디렉토리입니다.
  - com/example/: 패키지 구조입니다. 일반적으로 도메인 이름을 거꾸로 사용합니다.
  - controller/: 서블릿 파일을 저장하는 디렉토리입니다.
  - model/: 비즈니스 로직과 데이터베이스 모델을 저장하는 디렉토리입니다.
  - service/: 서비스 로직을 저장하는 디렉토리입니다.
  - resources/: 설정 파일과 기타 리소스 파일을 저장하는 디렉토리입니다.
  - log4j.properties: 로그 설정 파일입니다.

Web Project 구조

- **Java Maven 프로젝트의 기본 구조**
  - webapp/: 웹 애플리케이션 관련 파일을 저장하는 디렉토리입니다.
  - WEB-INF/: 웹 애플리케이션의 보안 설정 파일을 저장하는 디렉토리입니다.
  - web.xml: 웹 애플리케이션 배포 설명자 파일입니다.
  - jsp/: JSP 파일을 저장하는 디렉토리입니다.
  - lib/: 의존 라이브러리 파일을 저장하는 디렉토리입니다.
  - static/: 정적 파일(CSS, JavaScript, 이미지 등)을 저장하는 디렉토리입니다.
  - css/: CSS 파일을 저장하는 디렉토리입니다.
  - js/: JavaScript 파일을 저장하는 디렉토리입니다.
  - images/: 이미지 파일을 저장하는 디렉토리입니다.
  - pom.xml: Maven 프로젝트의 메타데이터와 의존성 정보를 담고 있는 파일입니다.

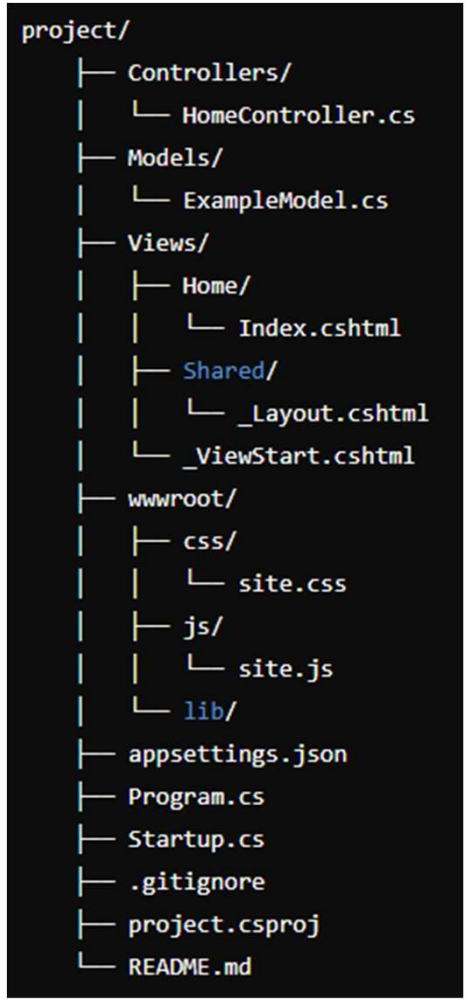
Web Project 구조

- **ASP.NET Core MVC 프로젝트 디렉토리 구조**
  - 루트 디렉토리 (project/)
  - Controllers/: 요청을 처리하고 응답을 반환하는 컨트롤러 파일을 저장하는 디렉토리입니다.
  - Models/: 애플리케이션의 데이터 모델을 정의하는 파일을 저장하는 디렉토리입니다.
  - Views/: 뷰 파일을 저장하는 디렉토리입니다. Razor 뷰 파일(.cshtml)을 포함합니다.
  - Home/: 특정 컨트롤러에 대한 뷰 파일을 저장하는 디렉토리입니다.
  - Shared/: 공통 레이아웃과 부분 뷰 파일을 저장하는 디렉토리입니다.
  - _Layout.cshtml: 기본 레이아웃 파일입니다.
  - _ViewStart.cshtml: 각 뷰가 렌더링될 때 먼저 실행되는 코드가 포함된 파일입니다.
  - wwwroot/: 정적 파일(CSS, JavaScript, 이미지 등)을 저장하는 디렉토리입니다.

```
project/
    ├── Controllers/
    │   └── HomeController.cs
    ├── Models/
    │   └── ExampleModel.cs
    ├── Views/
    │   ├── Home/
    │   │   └── Index.cshtml
    │   ├── Shared/
    │   │   └── _Layout.cshtml
    │   └── _ViewStart.cshtml
    ├── wwwroot/
    │   ├── css/
    │   │   └── site.css
    │   ├── js/
    │   │   └── site.js
    │   └── lib/
    ├── appsettings.json
    ├── Program.cs
    ├── Startup.cs
    ├── .gitignore
    ├── project.csproj
    └── README.md
```

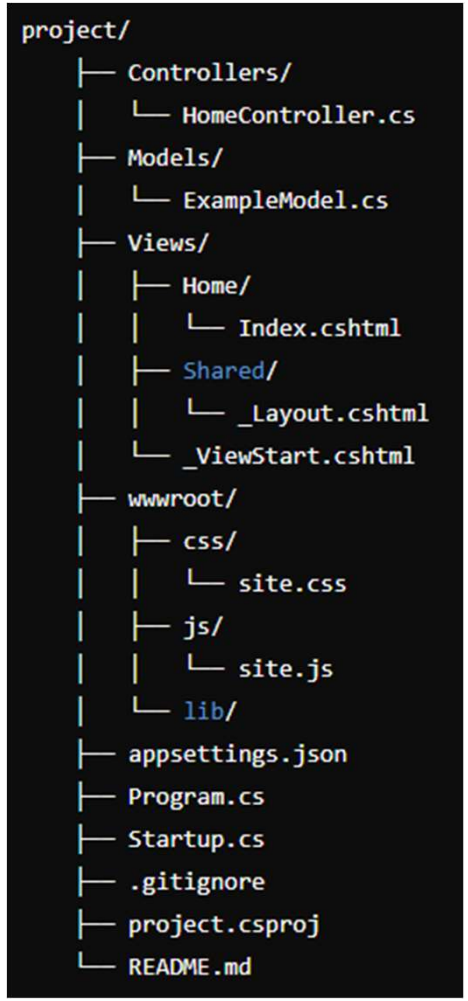
Web Project 구조

- **ASP.NET Core MVC 프로젝트 디렉토리 구조**
  - css/: CSS 파일을 저장하는 디렉토리입니다.
  - site.css: 기본 CSS 파일입니다.
  - js/: JavaScript 파일을 저장하는 디렉토리입니다
  - site.js: 기본 JavaScript 파일입니다.
  - lib/: 클라이언트 라이브러리(NPM, Bower 등)를 저장하는 디렉토리입니다.
  - appsettings.json: 애플리케이션 설정 파일입니다. 데이터베이스 연결 정보, 애플리케이션 설정 등을 포함합니다.
  - Program.cs: 애플리케이션의 진입점 파일입니다. ASP.NET Core 호스트를 구성하고 실행합니다.
  - Startup.cs: 애플리케이션을 설정하고 미들웨어 파이프라인을 구성하는 파일입니다.



Web Project 구조

- **ASP.NET Core MVC 프로젝트 디렉토리 구조**
  - .gitignore: Git이 추적하지 않을 파일과 디렉토리를 지정하는 파일입니다.
  - project.csproj: 프로젝트 파일로, 프로젝트의 메타데이터와 의존성 정보를 담고 있습니다.
  - README.md: 프로젝트에 대한 설명, 설치 방법, 사용 방법 등을 담고 있는 문서 파일입니다.



Web Project 구조

- **브라우저의 개발자 도구중 유용한 도구 정리할 것**

  - 브라우저의 개발자 도구중 유용한 도구 정리할 것