

# 미션 1 - TypeScript를 활용하여 ToDo List 만들기!

## 설계

BEM 방법론에 따라 작성해야 한다.

html 클래스명을 작성할 때는 `-` 를 통해 작성한다. (ex. `todo-container` )

## UI 구조 분석



하나의 블록( `todo-container` ) 안에 3개의 element로 구성

### Element

#### ▼ todo-container\_\_header

##### 내부 Element

- `todo-container__title` : header 선에서 작성 가능


#### ▼ todo-container\_\_form

##### 내부 Element

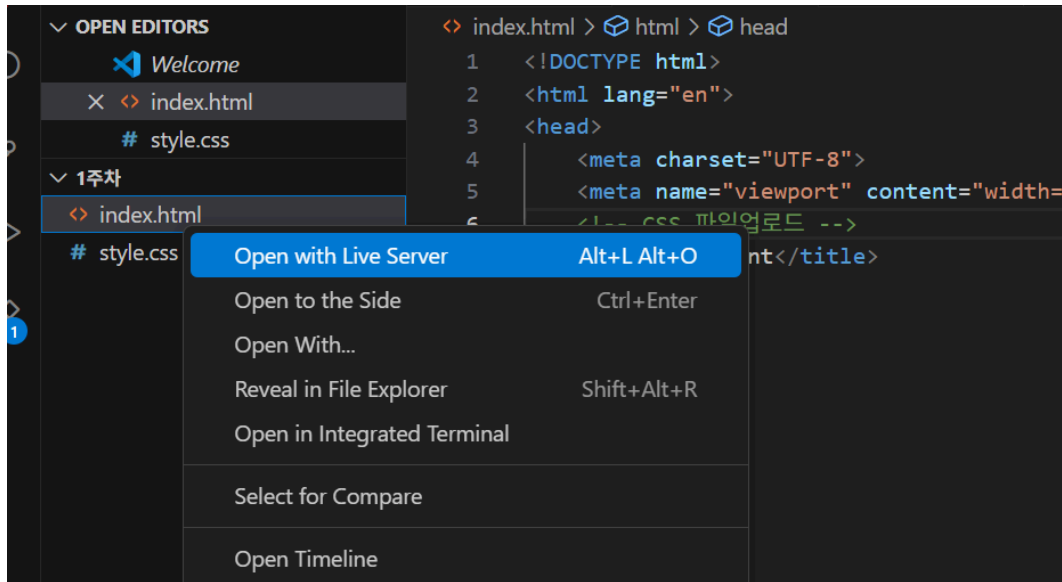
- `todo-container__text-field`

- todo-container\_\_add-button
- ▼ todo-container\_\_list
  - 내부 Element**
  - ▼ todo-container\_\_todo-list
    - 내부 Element**
    - todo-container\_\_todo-list-title
    - ▼ todo-container\_\_todo-list-element
      - todo-container\_\_todo-list-element-text
      - todo-container\_\_todo-list-element-button
  - ▼ todo-container\_\_done-list
    - 내부 Element'**
    - todo-container\_\_done-list-title
    - ▼ todo-container\_\_done-list-element
      - todo-container\_\_done-list-element-text
      - todo-container\_\_done-list-element-button

## HTML

index.html →  누르면 기본 구조 생성

Live Server을 설치하면 html에서 바로 인터넷 창을 연결할 수 있다.



### ▼ CSS와 연결?

`Ctrl + /` : vsc 주석 → CSS 파일 업로드 작성

`link:css` 작성하고 `Tab` 키

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- CSS 파일업로드 -->
  <link rel="stylesheet" href="style.css">
  <title>Document</title>
</head>
<body>

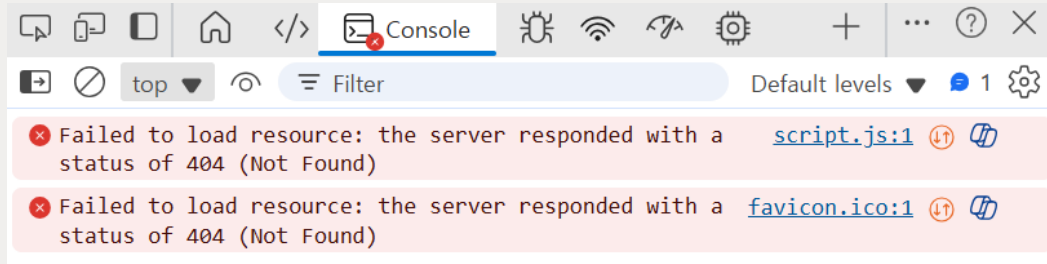
</body>
</html>
```

### ▼ TypeScript와 연결?

마찬가지로 주석 달고 `link:module`

```
<!-- JS 파일업로드 -->
<script type="module" src="./dist/script.js" defer></script>
```

⚠ 만약 실행이 잘 안되는 경우



터미널에 `tsc --watch` → dist 파일이 생기고 JS 파일도 생김

`Shift + Alt + F` : 자동정렬기능

#### ▼ html(초안)

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- CSS 파일업로드 -->
  <link rel="stylesheet" href="style.css">
  <!-- JS 파일업로드 -->
  <script type="module" src="./dist/script.js" defer></script>
  <title>Document</title>
</head>

<body>
```

```

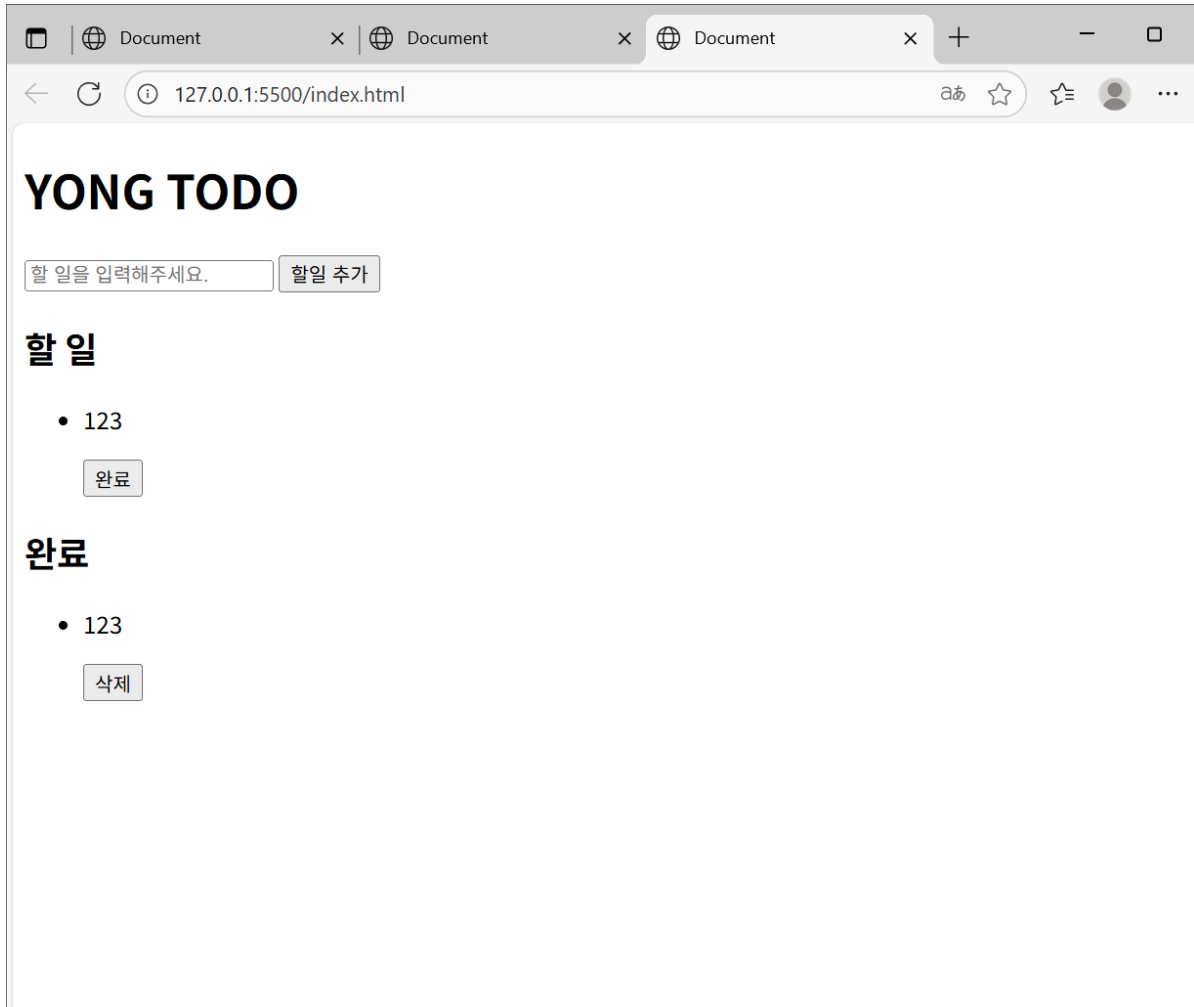
<div class="todo-container">
  <h1 class="todo-container__header">YONG TODO</h1>
  <form class="todo-container__form">
    <input type="text" , class="todo-container__text-field" , placeholder="할 일을 입력해주세요." , required>
    <button type="submit" , class="todo-container__add-button">할일
추가</button>
  </form>
  <div class="todo-container__list">
    <div class="todo-container__todo-list">
      <h2 class="todo-container__todo-list-title">할 일</h2>
      <ul>
        <li class="todo-container__done-list-element">
          <p class="todo-container__done-list-element-text">123</p>
        <button class="todo-container__done-list-element-button">완료</button>
      </li>
    </ul>
  </div>
  <div class="todo-container__done-list">
    <h2 class="todo-container__done-list-title">완료</h2>
    <ul>
      <li class="todo-container__done-list-element">
        <p class="todo-container__done-list-element-text">123</p>
      <button class="todo-container__done-list-element-button">삭제</button>
    </li>
  </ul>
  </div>
</div>
</body>

```

```
</html>
```

구조도 커스텀하다 보니 강의랑 조금 다르다

특히 `todo-container_done-list` 쪽 `ul` 부분 class를 어떻게 설정할지 모르겠다.



▼ TypeScript 연결을 위한 일부 요소 id 추가

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.
```

```

0">
  <!-- CSS 파일업로드 -->
  <link rel="stylesheet" href="style.css">
  <!-- JS 파일업로드 -->
  <script type="module" src="./dist/script.js" defer></script>
  <title>Document</title>
</head>

<body>
  <div class="todo-container">
    <h1 class="todo-container__header">YONG TODO</h1>
    <form id="todo-form", class="todo-container__form">
      <input id="todo-input", type="text", class="todo-container__text-
field", placeholder="할 일을 입력해주세요.", required>
      <button type="submit", class="todo-container__add-button">할일
추가</button>
    </form>
    <div class="todo-container__list">
      <div class="todo-container__todo-list">
        <h2 class="todo-container__todo-list-title">할 일</h2>
        <ul id="todo-list">
          <li class="todo-container__todo-list-element">
            <p class="todo-container__todo-list-element-text">123</p
>
            <button class="todo-container__todo-list-element-button">
완료</button>
          </li>
        </ul>
      </div>
      <div class="todo-container__done-list">
        <h2 class="todo-container__done-list-title">완료</h2>
        <ul id="done-list">
          <li class="todo-container__done-list-element">
            <p class="todo-container__done-list-element-text">123</p
>
            <button class="todo-container__done-list-element-butto

```

```

n">삭제</button>
    </li>
  </ul>
</div>
</div>
</div>
</body>

</html>

```

## CSS

style.css, BEM 방법론을 통해 작성한다.

html(초안)을 가지고 CSS로 형태를 짠다.

### ▼ 전체 스타일 잡기 : \*

```

*{
  margin: 0; /*바깥 여백 제거*/
  padding: 0; /*안쪽 여백 제거*/
  box-sizing: border-box; /*width×height 크기 보장*/
}

```

모든 HTML 요소에 공통 스타일을 적용한다는 뜻

### ▼ html body css

```

body{
  font-family: 'Roboto', sans-serif; /*폰트 설정*/
  display: flex; /*자식 요소들을 row 방향 정렬*/
  justify-content: center; /*가로 축 정렬*/
  align-items: center; /*세로 축 정렬*/
  height: 100vh; /*높이가 전체를 차지 못했기 때문에 정렬을 위해 높이 늘리기*/
  background-color: rgb(232, 231, 231);
}

```



`display: flex` 를 하면 `body` 컨테이너에 있는 직계 자식 요소가 `flex_item` 이 된다.

#### ▼ todo-container

```
.todo-container{  
  background: white;  
  padding: 20px;  
  border-radius: 12px; /*모서리 둥글게*/  
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); /*그림자 효과*/  
  width: 350;  
  text-align: center; /*텍스트 가운데 정렬*/  
}
```

#### ▼ todo-container\_\_header

```
.todo-container__header{  
  font-size: 24px;  
  margin-bottom: 16px; /*아래쪽 바깥 여백 늘리기*/  
}
```

#### ▼ todo-container\_\_form

```
.todo-container__form{  
  display: true;  
  gap: 10px;  
  margin-bottom: 20px;  
}
```

#### ▼ todo-container\_\_text-field

```
.todo-container__text-field{  
  flex:1; /*해당 자식 요소가 버튼 크기 제외 남는 공간을 전부 차지하도록*/  
  padding: 8px;  
  border: 1px solid #ccc; /*요소 테두리를 설정*/  
  border-radius: 6px;
```

```
font-size: 14px;
}
```

#### ▼ todo-container\_\_add-button

```
.todo-container__add-button{
  background-color: rgb(124, 224, 88);
  color: white;
  border: none;
  padding: 8px 12px;
  cursor: pointer; /*커서를 대면 눌릴 수 있도록*/
  border-radius: 6px;
  transition: background-color 0.3s ease; /*배경색이 바뀌면 부드럽게
  전환*/
}

/*hover : 마우스가 감지되면 다른 스타일 적용*/
.todo-container__add-button:hover{
  background-color: green;
}
```

#### ▼ todo-container\_\_list

```
.todo-container__list{
  display: flex;
  justify-content: space-between; /*flex_item을 양끝에*/
  gap: 15px; /*flex_item의 간격*/
}
```

#### ▼ todo-container\_\_todo-list

```
.todo-container__todo-list{
  width: 100%;
  text-align: left;
}
```

▼ todo-container\_\_todo-list-title

```
.todo-container__todo-list-title{  
  font-size: 18px;  
  margin-bottom: 10px;  
  justify-content: center;  
}
```

▼ todo-container\_\_todo-list-element

```
.todo-container__todo-list-element{  
  list-style: none; /*unordered list라 나오는 점을 지울 수 있음*/  
  display: flex;  
}
```

▼ todo-container\_\_todo-list-element-text

```
.todo-container__todo-list-element-text{  
  flex:1;  
  padding: 8px;  
  border: 1px solid #ddd;  
  border-radius: 6px;  
  margin-bottom: 6px;  
  width: 100%;  
}
```

▼ todo-container\_\_todo-list-element-button

```
.todo-container__todo-list-element-button{  
  background-color: rgb(124, 224, 88);  
  color: white;  
  border: none;  
  padding: 8px 12px;  
  cursor: pointer;  
  border-radius: 6px;
```

```

font-size: 12px;
transition: background-color 0.3s ease;
margin-bottom: 6px;
}

.todo-container__todo-list-element-button:hover{
background-color: green;
}

```

#### ▼ todo-container\_\_done-list

```

.todo-container__done-list{
width: 100%;
text-align: left;
}

```

#### ▼ todo-container\_\_done-list-title

```

.todo-container__done-list-title{
font-size: 18px;
margin-bottom: 10px;
justify-content: center;
display: flex;
}

```

#### ▼ todo-container\_\_done-list-element

```

.todo-container__done-list-element-button{
background-color: red;
color: white;
border: none;
padding: 8px 12px;
cursor: pointer;
border-radius: 6px;
font-size: 12px;
}

```

```

transition: background-color 0.3s ease;
margin-bottom: 6px;
}

.todo-container__done-list-element-button:hover{
background-color: rgb(183, 0, 0);
}

```

## TypeScript

```

"rootDir": "./src", // 소스 파일의 루트 디렉토리
"outDir": "./dist", // 컴파일된 파일이 저장될 디렉토리

```

`tsconfig.json`에 이렇게 정의되었기 때문에

`src` 폴더에 `scrip.ts` 파일을 만든다. JS 파일은 `dist` 폴더에 생성된다.

실행은 `npx tsc --watch`

### ▼ 코드 원문

```

const todoInput = document.getElementById("todo-input") as HTMLInputElement;
const todoForm = document.getElementById("todo-form") as HTMLFormElement;
const todoList = document.getElementById("todo-list") as HTMLUListElement;
const doneList = document.getElementById("done-list") as HTMLUListElement;

type Todo = {
  id: number
  text: string
}

let todos: Todo[] = [];
let doneTasks: Todo[] = [];

```

```

const renderTask = (): void =>{
  todoList.innerHTML = '';
  doneList.innerHTML = '';

  todos.forEach((todo) : void => {
    const li = createTodoElement(todo, false);
    todoList.appendChild(li);
  });

  doneTasks.forEach((todo): void =>{
    const li = createTodoElement(todo, true)
    doneList.appendChild(li);
  })
}

const getTodoText = ():string =>{
  return todoInput.value.trim();
}

const addTodo = (text: string): void =>{
  todos.push({id : Date.now(), text: text});
  //입력창 초기화
  todoInput.value = '';
  renderTask();
}

const completeTodo = (todo: Todo) : void => {
  todos = todos.filter((x): Boolean => x.id !== todo.id)
  doneTasks.push(todo);
  renderTask();
}

const deleteTodo = (todo: Todo): void => {
  doneTasks = doneTasks.filter((x): Boolean => x.id !== todo.id)
  renderTask();
}

```

```

}

/*
동적으로 어떻게 만들 수 있을까?
<ul id="todo-list">
  <li class="todo-container__todo-list-element">
    <p class="todo-container__todo-list-element-text">123</p>
    <button class="todo-container__todo-list-element-button">완료</butt
on>
  </li>
</ul>
*/

//완료되면 true, 안되면 false
const createTodoElement = (todo: Todo, isDone: boolean) : HTMLElement
⇒ {
  //<li> 만들기
  const li = document.createElement('li');
  li.classList.add('todo-container__todo-list-element');

  //<p> 만들기
  const p = document.createElement('p');
  p.classList.add('todo-container__todo-list-element-text');
  p.textContent = todo.text;

  //<button> 만들기
  const button = document.createElement('button');
  button.classList.add('todo-container__todo-list-element-button');

  if (isDone){
    button.textContent = '삭제';
    button.style.backgroundColor = 'red';
  } else{
    button.textContent = '완료';
    button.style.backgroundColor = 'rgb(124, 224, 88)'
  }
}

```

```

button.addEventListener('click', ():void =>{
  if (isDone){
    deleteTodo(todo);
  } else{
    completeTodo(todo);
  }
})

//Node : html의 요소
//appendChild는 Node를 인자로 받음
li.appendChild(p)
li.appendChild(button)
return li;
}

todoForm.addEventListener('submit', (event: Event):void =>{
  event.preventDefault();
  const text = getTodoText();
  if (text){
    addTodo(text);
  }
})

```

## HTML 요소 선택

```

const todoInput = document.getElementById("todo-input") as HTMLInputElement;
const todoForm = document.getElementById("todo-form") as HTMLFormElement;
const todoList = document.getElementById("todo-list") as HTMLUListElement;
const doneList = document.getElementById("done-list") as HTMLUListElement;

```

`todoForm` 이 필요한 이유는 이벤트를 처리를 위해, `<form>` 태그는 입력값을 서버로 제출하기 때문



## 할 일에 대한 Type 정의

```
type Todo = {  
  id: number  
  text: string  
}  
let todos: Todo[] = [];  
let doneTasks: Todo[] = [];
```

## 배열과 화면(DOM) 동기화

```
const renderTask = (): void =>{  
  todoList.innerHTML = '';  
  doneList.innerHTML = '';  
}
```

`todos` 를 반영하기 전 기존 화면을 지워 초기화

실제론 `<ul>` 태그는 남고 `<li>` 들이 전부 날라감

## +아이템 생성 함수 도입

```
const renderTask = (): void =>{  
  todoList.innerHTML = '';  
  doneList.innerHTML = '';  
  
  todos.forEach((todo) : void => {  
    const li = createTodoElement(todo, false);  
    todoList.appendChild(li);  
  });  
  
  doneTasks.forEach((todo): void =>{  
    const li = createTodoElement(todo, true)  
    doneList.appendChild(li);  
  })  
}
```

`createTodoElement` 함수를 통해 `<li>` 요소들을 생성해 `todoList` / `doneList`에 추가

## 함수

### 할 일 텍스트 처리

```
const getTodoText = ():string =>{
  return todoInput.value.trim();
};
```

### 할 일 추가 처리

```
const addTodo = (text: string): void =>{
  todos.push({id : Date.now(), text: text});
  //입력창 초기화
  todoInput.value = "";
  renderTask();
}
```

### 할 일 상태 변경

```
const completeTodo = (todo: Todo) : void => {
  todos = todos.filter((x): Boolean => x.id !== todo.id)
  doneTasks.push(todo);
  renderTask();
}
```

### 완료한 일 삭제

```
const deleteTodo = (todo: Todo): void => {
  doneTasks = doneTasks.filter((x): Boolean => x.id !== todo.id)
  renderTask();
}
```

## 할 일 아이템 생성 함수

```

/*
동적으로 어떻게 만들 수 있을까?
<ul id="todo-list">
  <li class="todo-container__todo-list-element">
    <p class="todo-container__todo-list-element-text">123</p>
    <button class="todo-container__todo-list-element-button">완료</button>
  </li>
</ul>
*/

//완료되면 true, 안되면 false
const createTodoElement = (todo: Todo, isDone: boolean) : HTMLElement => {
  //<li> 만들기
  const li = document.createElement('li');
  li.classList.add('todo-container__todo-list-element');

  //<p> 만들기
  const p = document.createElement('p');
  p.classList.add('todo-container__todo-list-element-text');
  p.textContent = todo.text;

  //<button> 만들기
  const button = document.createElement('button');
  button.classList.add('todo-container__todo-list-element-button');

  if (isDone){
    button.textContent = '삭제';
    button.style.backgroundColor = 'red';
  } else{
    button.textContent = '완료';
    button.style.backgroundColor = 'rgb(124, 224, 88)'
  }

  button.addEventListener('click', ():void =>{
    if (isDone){

```

```

        deleteTodo(todo);
    } else{
        completeTodo(todo);
    }
})

//Node : html의 요소
//appendChild는 Node를 인자로 받음
li.appendChild(p)
li.appendChild(button)
return li;
}

```

`createTodoElement` : 동적으로 html 요소를 만드는 함수. `HTMLElement` 을 반환해 `todoList ( <ul> )` 에 추가

나 같은 경우는 `<li>`를 `<p>`와 `<button>`으로 만들었기 때문에 각각 요소를 만든 다음 `appendChild` 로 추가했다.

## 폼 제출 이벤트 리스터

```

todoForm.addEventListener('submit', (event: Event):void =>{
    event.preventDefault();
    const text = getTodoText();
    if (text){
        addTodo(text);
    }
})

```