


REACT -

목차

>4/17일	2
Bootstrap 5의 마진, 패딩, 전경색, 배경색 사용법	2
1. 마진(Margin)과 패딩(Padding)	2
기본 구조	2
예시 코드	2
사이즈 단계	2
2. 배경색(Background Color)	3
기본 클래스	3
배경 그라데이션 추가	3
3. 전경색(Text Color)	4
기본 클래스	4
불투명도 조절	4
4. 응용 예제	4
>3/26일	5
리액트의 정석 with 타입스크립트	5
>4/11일	7
설명:	8
예제 코드:	8
 3. React-Bootstrap <Form> 사용법	9
◆ React-Bootstrap 기본 폼 예제	9
◆ React-Bootstrap의 폼 요소	10
>4/12일	15
>4/13일	18
>4/17일	20

>4/17일

Bootstrap 5의 마진, 패딩, 전경색, 배경색 사용법

1. 마진(Margin)과 패딩(Padding)

기본 구조

- `{property}{sides}-{size}` 형식
- property: `m`(margin), `p`(padding)
- sides: `t`(top), `b`(bottom), `s`(start/left), `e`(end/right), `x`(좌우), `y`(상하), 생략시(4방향)
- size: `0~5`, `auto`

예시 코드

html

Copy

Download

Run

```
<div class="mt-3">상단 마진 1rem</div>
<div class="pb-4">하단 패딩 1.5rem</div>
<div class="mx-auto">좌우 마진 자동(가운데 정렬)</div>
<div class="p-3">4방향 패딩 1rem</div>
```

사이즈 단계

클래스	rem 값	px 값
-0	0	0
-1	0.25	4
-2	0.5	8

-3	1	16
-4	1.5	24
-5	3	48

2. 배경색(Background Color)

기본 클래스

html

Copy

Download

Run

```
<div class="bg-primary">기본 주요 색상</div>
```

```
<div class="bg-secondary">보조 색상</div>
```

```
<div class="bg-success">성공 색상</div>
```

```
<div class="bg-danger">위험/오류 색상</div>
```

```
<div class="bg-warning">경고 색상</div>
```

```
<div class="bg-info">정보 색상</div>
```

```
<div class="bg-light">밝은 색상</div>
```

```
<div class="bg-dark">어두운 색상</div>
```

```
<div class="bg-white">흰색</div>
```

```
<div class="bg-transparent">투명</div>
```

배경 그라데이션 추가

html

Copy

Download

Run

```
<div class="bg-primary bg-gradient">그라데이션 효과</div>
```

3. 전경색(Text Color)

기본 클래스

html

Copy

Download

Run

```
<p class="text-primary">기본 주요 색상</p>
<p class="text-secondary">보조 색상</p>
<p class="text-success">성공 색상</p>
<p class="text-danger">위험/오류 색상</p>
<p class="text-warning">경고 색상</p>
<p class="text-info">정보 색상</p>
<p class="text-light">밝은 색상(어두운 배경에)</p>
<p class="text-dark">어두운 색상(밝은 배경에)</p>
<p class="text-white">흰색</p>
<p class="text-body">이 내용은 문서의 기본 텍스트 색상으로 표시됩니다.</p>
<p class="text-muted">흐릿한 색상</p>
```

불투명도 조절

html

Copy

Download

Run

```
<p class="text-black-50">검정색 50% 불투명</p>
<p class="text-white-50">흰색 50% 불투명</p>
```

4. 응용 예제

html

Copy

Download

Run

```
<div class="bg-dark text-white p-4 m-3">
  <h2 class="text-warning">제목</h2>
  <p class="text-light">내용 내용 내용</p>
  <button class="btn btn-primary mt-3 ms-2">버튼</button>
</div>
```

이 코드는:

- 어두운 배경에 흰색 텍스트
- 내부에 1rem 패딩(p-4)
- 외부에 1rem 마진(m-3)
- 제목은 경고색(yellow)
- 내용은 밝은 회색
- 버튼은 상단 마진 1rem(mt-3), 왼쪽 마진 0.5rem(ms-2) 적용

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Bootstrap 예제</title>
  <!-- Bootstrap CSS CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <!-- Bootstrap JS Bundle (옵션) -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min
.js"></script>
</head>
<body class="bg-light p-4">

  <div class="mt-3 bg-white border p-2">
    상단 마진 1rem
  </div>

  <div class="pb-4 bg-white border p-2 mt-4">
    하단 패딩 1.5rem
  </div>

  <div class="mx-auto bg-white border p-2 mt-4 text-center" style="width:
50%;">
    좌우 마진 자동(가운데 정렬)
  </div>

  <div class="p-3 bg-white border mt-4">
```

```
    4방향 패딩 1rem  
</div>
```

```
</body>  
</html>
```

리액트의 정석 **with** 타입스크립트

Vite를 사용하는 이유

Vite는 기존의 웹 번들러(Webpack 등)보다 더 빠르고 효율적인 개발 환경을 제공하는 빌드 도구이다. 다음과 같은 특징을 통해 현대적인 프론트엔드 개발에서 널리 사용된다.

1. 빠른 개발 서버

Vite는 ES 모듈을 활용하여 전체 프로젝트를 번들링하지 않고 필요한 파일만 즉시 제공한다. 이를 통해 코드 변경 사항이 실시간으로 반영되는 **HMR(Hot Module Replacement)** 속도가 매우 빠르다.

2. 고속 빌드 성능

Vite는 **esbuild**를 사용하여 TypeScript, JSX 등을 빠르게 변환한다. **esbuild**는 기존 번들러보다 **10~100배** 빠른 변환 속도를 제공하며, 빌드 시간이 크게 단축된다.

3. 자동 코드 최적화

Vite는 개발 과정에서 자동으로 코드 스플리팅을 수행하고, 사용되지 않는 코드를 제거하는 **Tree Shaking**을 적용하여 번들 크기를 최소화한다.

4. 간편한 설정 및 사용 편의성

기존 번들러(Webpack 등)와 달리, Vite는 복잡한 설정이 필요하지 않다. 기본적으로 최적화된 환경을 제공하며, 추가적인 설정이 필요한 경우 **vite.config.js** 파일을 통해 간단히 조정할 수 있다.

5. 강력한 플러그인 시스템

Vite는 Rollup 기반의 플러그인 시스템을 사용하여 확장성이 뛰어나다. React, Vue, Svelte 등의 프레임워크를 손쉽게 설정할 수 있으며, PWA, SSR 등의 기능도 플러그인으로 추가할 수 있다.

6. SSR 및 PWA 지원

Vite는 서버 사이드 렌더링(**Server-Side Rendering, SSR**) 및 ****PWA(Progressive Web App)****를 위한 최적화 기능을 제공한다. 이를 통해 성능과 SEO(검색 엔진 최적화) 측면에서도 유리하다.

결론

Vite는 빠른 개발 서버, 고속 빌드 성능, 자동 최적화 등의 이점을 통해 기존 번들러보다 더 효율적이고 편리한 프론트엔드 개발 환경을 제공한다. React, Vue 등의 최신 프레임워크를 사용할 때 Vite를 선택하면 개발 생산성을 극대화할 수 있다.

프로젝트 세팅

1. `todolist`를 만든다.
2. `cd todolist` 폴더로 이동
3. `npm create vite@latest .` 설치 `react`과 `typescript`를 선택
4. `npm install`
5. `npm run dev` 실행

>4/11일

부트 스트랩 설치

```
npm install bootstrap
npm install react-bootstrap bootstrap
npm install bootstrap-icons
```

리액트 부트스트랩 사이트

<https://react-bootstrap.netlify.app/>

main.tsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

App.tsx

```
import './App.css'
import TodoList from './TodoList'
function App() {
  return (
    <>
      <div>
        <TodoList></TodoList>
      </div>
    </>
  )
}

export default App
```

TodoList.tsx

```
const TodoList: React.FC = () => { ... }
```

이 코드는 **React** 함수형 컴포넌트(**TodoList**)를 정의하는 코드입니다.

설명:

1. `const TodoList = () => { ... }`

- `TodoList`라는 변수를 화살표 함수로 정의하여 `React` 컴포넌트로 사용합니다.
- 함수형 컴포넌트는 `JSX`를 반환하는 역할을 합니다.

2. `: React.FC` (`React Functional Component` 타입)

- `React.FC`는 `FunctionComponent`의 약자로, 이 컴포넌트가 `React` 함수형 컴포넌트임을 명시하는 `TypeScript` 타입입니다.
- 이 타입을 사용하면 자동으로 `children`을 포함한 기본적인 타입 지원이 제공됩니다.

예제 코드:

3. React-Bootstrap <Form> 사용법

React-Bootstrap을 사용하면 더 스타일링된 폼을 쉽게 만들 수 있습니다.

◆ React-Bootstrap 기본 폼 예제

jsx

복사편집

```
import { useState } from "react";
import { Form, Button } from "react-bootstrap";

function BootstrapForm() {
  const [email, setEmail] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`입력한 이메일: ${email}`);
  };

  return (
    <Form onSubmit={handleSubmit}>
      <Form.Group controlId="email">
        <Form.Label>이메일 주소</Form.Label>
        <Form.Control
```

```

        type="email"
        placeholder="이메일 입력"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
    </Form.Group>
    <Button variant="primary" type="submit">제출</Button>
  </Form>
);
}

```

◆ **React-Bootstrap**의 폼 요소

- `<Form.Group>` → 입력 요소들을 그룹화
- `<Form.Label>` → 입력 필드의 라벨
- `<Form.Control>` → 입력 필드 (text, email, password 등 가능)
- `<Button>` → 제출 버튼

React.FC<TodoModalProps> → **TodoModal** 컴포넌트의 리턴 타입

TodoModalProps → props의 타입

```
import React, { useState } from "react";
import { Button, Container, Row, Col, Form } from 'react-bootstrap';

// 1. 타입 정의
type Todo = {
  id: number; // 할일의 고유 ID
  text: string; // 할일의 내용
  isChecked: boolean; // 할일 완료 여부
};

const TodoList: React.FC = () => {
  const title: string = '오늘 할일'; // 타이틀, 타입 추론으로 문자열로 설정됨

  // 2. 상태 관리
  const [todos, setTodos] = useState<Todo[]>([
    { id: 1, text: '잠자기', isChecked: false },
    { id: 2, text: '공부하기', isChecked: false },
    { id: 3, text: '밥먹기', isChecked: false },
    { id: 4, text: '산책하기', isChecked: false },
  ]);

  const [newTodo, setNewTodo] = useState<string>(''); // 새 할일 텍스트 입력 상태
  const [showDetail, setShowDetail] = useState<boolean>(false); // 상세보기 여부 상태
  const [selectedTodo, setSelectedTodo] = useState<Todo | null>(null); // 선택된 할일 상태

  // 3. 체크박스 상태 변경 처리
  const handleCheckboxChange = (itemId: number) => {
    setTodos((prevItems) =>
      prevItems.map((item) =>
        item.id === itemId ? { ...item, isChecked: !item.isChecked } : item
      )
    );
  };

  // 4. 새로운 할일 추가 처리
  const addTodo = () => {
    if (newTodo.trim() !== '') {
      setTodos([

```

```

        ...todos,
        { id: Date.now(), text: newTodo, isChecked: false }
    ]);
    setNewTodo(''); // 새 할일 입력 후 입력 필드를 비움
  }
}

// 5. 할일 삭제 처리
const removeTodo = (id: number) => {
  setTodos(todos.filter((todo) => todo.id !== id)); // 해당 ID를 제외한 나머지
  할일로 필터링
}

// 6. 할일 클릭 시 상세보기 열기
const handleTodoClick = (todo: Todo) => {
  setShowDetail(true);
  setSelectedTodo(todo); // 선택된 할일을 상태에 저장
}

// 7. 상세보기 닫기
const handleCloseDetail = () => {
  setShowDetail(false);
}

return (
  <Container fluid className="mt-5" style={{ maxWidth: '1200px' }}>
    {/* 제목 */}
    <Container className="p-3 mb-4 bg-dark text-white rounded">
      {/* p-3: 패딩, mb-4: 하단 마진, bg-dark: 어두운 배경색, text-white: 흰색
      텍스트, rounded: 둥근 모서리 */}
      <h1 className="text-center">{title}</h1>
    </Container>

    {/* 할일 입력 폼 */}
    <Row className="justify-content-center mb-4">
      <Col xs={10} md={8} lg={6}>
        <div className="border p-3 rounded mb-4 shadow-sm">
          {/* border: 테두리, p-3: 패딩, rounded: 둥근 모서리, mb-4: 하단
          마진, shadow-sm: 작은 그림자 */}
          <Form className="d-flex">
            {/* d-flex: Flexbox 레이아웃, 자식 요소들이 가로로 정렬됨
            */}

            <Form.Group controlId="newTodo" className="flex-grow-1
            me-2">

              {/* flex-grow-1: 가능한 공간을 채우도록 설정, me-2:
              오른쪽 마진 */}

```

```

        <Form.Control
            type="text"
            placeholder="할일 입력"
            value={newTodo} // 입력 값은 newTodo 상태에 바인딩
            onChange={(e) => setNewTodo(e.target.value)} //
입력 값 변경 시 상태 업데이트

            size="lg"
        />
    </Form.Group>
    <Button variant="warning" onClick={addTodo}
size="lg">추가</Button>
</Form>
</div>

{/* 할일 목록 출력 */}
<div className="border p-3 rounded shadow-sm">
    {/* border: 테두리, p-3: 패딩, rounded: 둥근 모서리, shadow-sm:
작은 그림자 */}

    <ul className="list-unstyled">
        {/* list-unstyled: 기본 목록 스타일 제거 */}
        {
            todos.map((todo) => (
                <li key={todo.id} className="d-flex
align-items-center justify-content-between mb-3" style={{ fontSize: '1.25rem' }}>
                    {/* d-flex: Flexbox 레이아웃,
align-items-center: 수직 중앙 정렬, justify-content-between: 양쪽 끝 정렬, mb-3: 하단
마진 */}

                    {/* 체크박스과 할일 텍스트 */}
                    <Form.Check
                        type="checkbox"
                        checked={todo.isChecked}
                        onChange={() =>
handleCheckboxChange(todo.id)}

                        label={todo.isChecked ?
<del>{todo.text}</del> : <span onClick={() =>
handleTodoClick(todo)}>{todo.text}</span>}

                        style={{ fontSize: '1.25rem' }}
                    />
                    {/* 삭제 버튼 */}
                    <Button variant="danger" size="lg" onClick={()
=> removeTodo(todo.id)} style={{ fontSize: '1rem' }}>
                        삭제
                    </Button>
                </li>
            ))
        }
    </ul>

```



```

        </ul>
      </div>
    </Col>
  </Row>

  { /* 날씨 및 시간 정보 */ }
  <Row className="justify-content-center mt-3">
    <Col xs={10} md={8} lg={6}>
      <div className="border p-3 rounded shadow-sm">
        { /* border: 테두리, p-3: 패딩, rounded: 둥근 모서리, shadow-sm:
작은 그림자 */ }

        <h4 className="mb-3">날씨 예보</h4>
        { /* mb-3: 하단 마진 */ }
        <p style={{ fontSize: '1.25rem', fontWeight: 'bold', color:
'#007bff' }}>

          현재 날씨: 맑음, 25°C

        </p>
        <h4 className="mt-4 mb-3">현재 시간</h4>
        { /* mt-4: 상단 마진, mb-3: 하단 마진 */ }
        <p style={{ fontSize: '1.25rem', fontWeight: 'bold', color:
'#007bff' }}>

          {new Date().toLocaleTimeString()}

        </p>
      </div>
    </Col>
  </Row>
</Container>
)
}

export default TodoList;

```

>4/12일

타이머 추가

```
/* 날씨 및 시간 정보 */
<Row className="justify-content-center mt-3">
  <Col xs={10} md={8} lg={6}>
    <div className="border p-3 rounded shadow-sm">
      /* border: 테두리, p-3: 패딩, rounded: 둥근 모서리,
shadow-sm: 작은 그림자 */
      <h4 className="mb-3">날씨 예보</h4>
      /* mb-3: 하단 마진 */
      <p style={{ fontSize: '1.25rem', fontWeight: 'bold',
color: '#007bff' }}>
        현재 날씨: 맑음, 25°C
      </p>
      <h4 className="mt-4 mb-3">현재 시간</h4>
      /* mt-4: 상단 마진, mb-3: 하단 마진 */
      <p style={{ fontSize: '1.25rem', fontWeight: 'bold',
color: '#007bff' }}>
        {new Date().toLocaleTimeString()}
      </p>
    </div>
  </Col>
</Row>

import { useState } from "react"
import { Button } from "react-bootstrap";
const Timer : React.FC = () => {
  const [seconds, setSeconds] = useState<number>(0);

  return(
    <div>
```

```

    <h4>타이머 : {seconds} 초</h4>
    <Button onClick={
      ()=>{
        setInterval(()=>{
          setSeconds((prevSeconds)=> {
            return prevSeconds + 1
          })
        }, 1000);
      }
    }>시작</Button>
  </div>
)
}

export default Timer;

```

>4/13일

```
import React from "react"

import {Button, Modal} from 'react-bootstrap';

type Todo = {
  id : number;
  text : string;
  isChecked : boolean;
};

type TodoModalProps = {
  show : boolean;
  handleClose : () => void;
  todo : Todo | null;
}

const TodoModal:React.FC<TodoModalProps> = ({show, handleClose, todo})=>{
  return(
    <Modal show={show} onHide = {()=>{handleClose()}} centered>
      <Modal.Header closeButton>
        <Modal.Title>상세정보</Modal.Title>
      </Modal>
    )
  }
```

```

        </Modal.Header>

        <Modal.Body>

            {todo?.text}의 자세한 정보를 출력합니다.

            <p>현재날짜 : {new Date().toLocaleDateString()}</p>

        </Modal.Body>

    </Modal>

    )
}

export default TodoModal;

```

TODOLIST 프로젝트에서

INDEX.CSS파일에 BODDY부분을 주석하면 썸그라운드 화면이 펼쳐집니다.

```

body {

    margin: 0;

    display: flex;

    place-items: center;

    min-width: 320px;

    min-height: 100vh;

```

}

>4/17일
