리액트 스터디 4강

캡스 39기 송윤석

강의 전체 로드맵

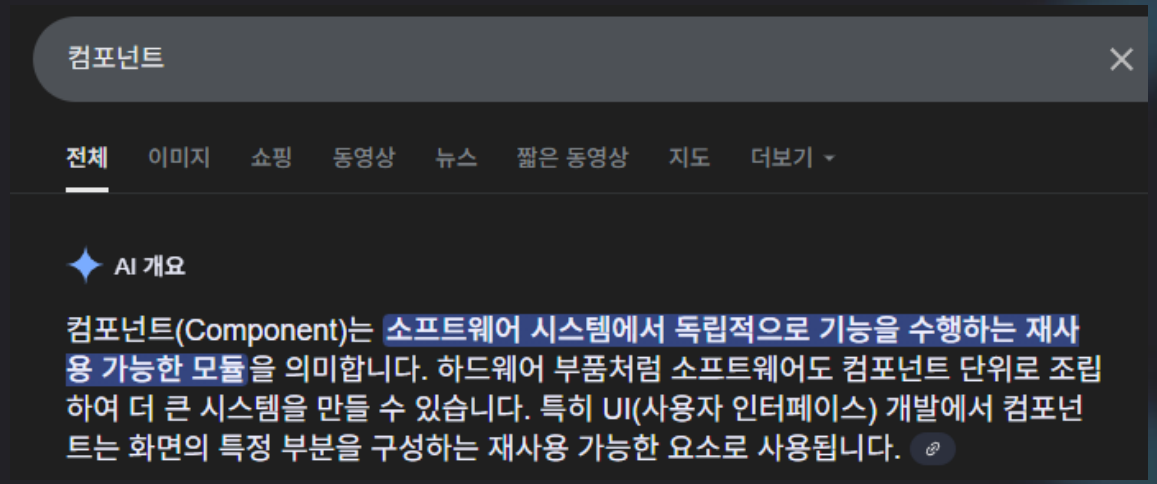
- 1강: 스터디 OT, 개발환경 구축, HTML/CSS ✅
- 2강: Javascript, Typescript ✅
- 3강: React의 세계관, Electron 개념, 프로젝트 생성 ✅
- 4강: 컴포넌트, 상태관리, 라우팅 <- 오늘
- 5강: 훅, 전역상태관리
- 6강: 라이브러리, 배포

목차 및 오늘의 목표

- 4.1 컴포넌트 개념과 역할
- 4.2 JSX 문법 소개
- 4.3 Props를 통한 데이터 전달
- 4.4 useState로 상태 관리하기
- 4.5 이벤트 처리: onClick, onChange 등
- 4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)
- 4.7 React Router 개요 및 설치
- 4.8 Route, Link, useNavigate 활용
- 4.9 간단한 라우팅 실습 (페이지 분리)

4.1 컴포넌트 개념과 역할

- 컴포넌트란?



4.1 컴포넌트 개념과 역할

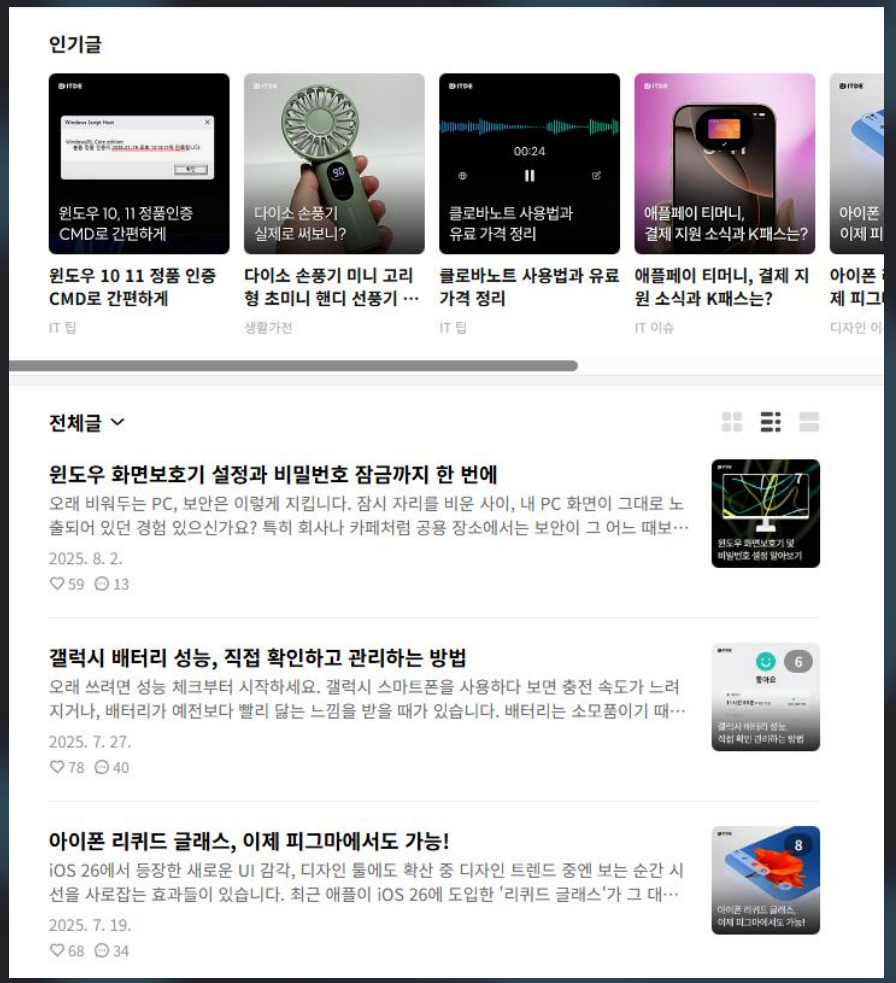
- **리액트(프론트엔드)에서 컴포넌트란?**

UI + 로직 + 상태를 하나로 묶은,
재사용 가능한 단위

- **즉, 웹페이지를 구성하는 작은 블록**

웹사이트는 비슷한 요소들이 반복되기 때문에
👉 **하나의 컴포넌트를 여러 번 재사용**하는 것이 효율적

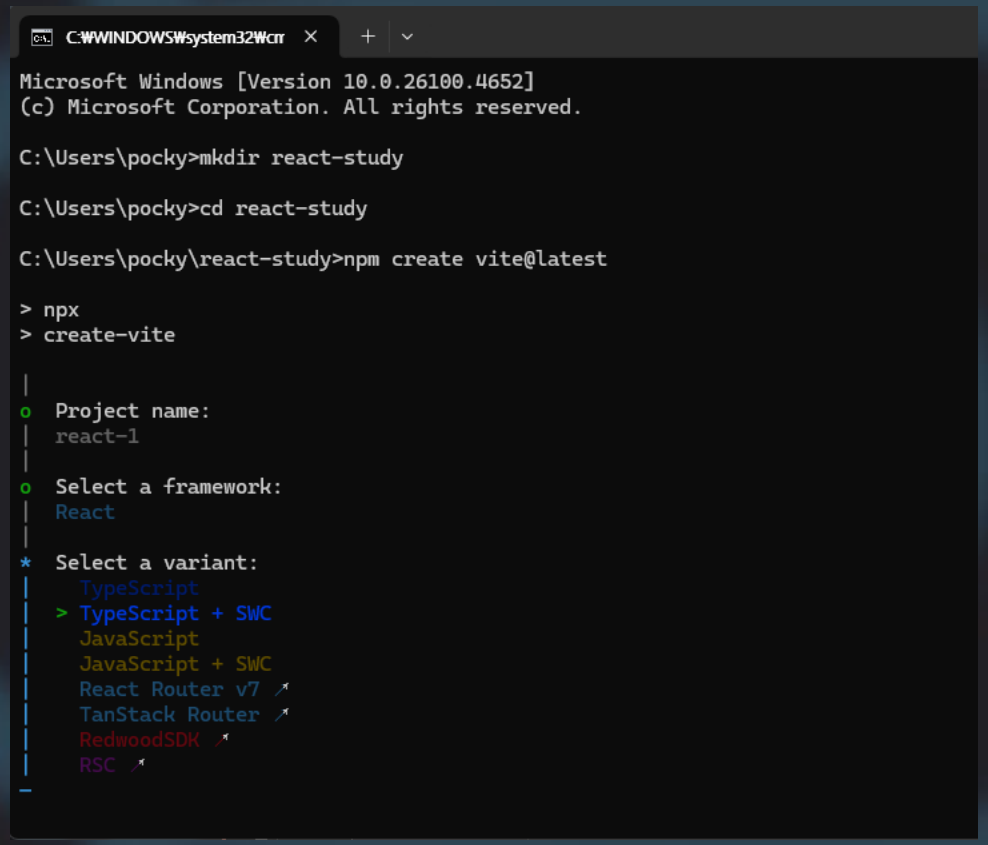
"하나 만들고, 여러 번 쓰는 게 컴포넌트의 핵심!"



예시사진 – 네이버 블로그

4.2 JSX 문법 소개

- **JSX를 배우기 전, (VSCode, Node.js도 설치 안되었다면...)**
- **리액트 프로젝트 세팅하기**
  - 윈도우 기준 **ctrl + R -> 명령창 열기**
  - **cmd를 입력 후, 명령 프롬프트에서 프로젝트 폴더 생성**
    - **mkdir 폴더이름**
  - **폴더 안으로 이동 후 npm create vite@latest**
    - **Framework – React**
    - **variant – Typescript + SWC 선택**
  - **설치 및 실행**
    - **npm install**
    - **npm run dev**



- **Vite는 무엇인가요?**
  - 개발 서버 실행기이자 **번들러**
  - 리액트의 JSX 문법을 브라우저가 이해할 수 있게 **JS 코드**로 변환
  - 개발 중에는 **빠르게 화면을 갱신**
  - 여러 JSX 파일들을 하나로 묶고, **최적화된 형태로 만듬**

4.2 JSX 문법 소개

- **JS(자바스크립트) + XML(<태그>값<태그>형식의 문법)**
- 자바스크립트 코드 안에 **HTML처럼 보이는 문법을 쓸 수 있게 해주는 리액트의 문법**

- **JSX의 목적**
  - **리액트에서 UI를 구성하기 위한 문법**
  - **JS 안에서 조건문, 반복문, 상태 등을 이용해 유연하게 HTML 생성 가능**
  - **가독성, 생산성 향상**

src > App.tsx > ...

```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

4.2 JSX 문법 소개

- JSX 템플릿 문법 - JS 코드, 값 넣기
- JSX 함수의 return 안 XML 문법에서 중괄호 { } 안에는 JS 코드가 들어갈 수 있음

```
function App() {
  const speak = "Hello, JSX!";
  return (
    <div>
      <h1>{speak}</h1>
      <p>This is a simple JSX practice component.</p>
    </div>
  );
}
```

4.2 JSX 문법 소개

- JSX 템플릿 문법 – 조건부 렌더링
- 특정 조건에서만 요소를 보여줘야 하는 경우가 있음
  - 1. 삼항 연산자 사용
  - 2. 논리 연산자 사용
  - 3. return을 여러 개

- 삼항 연산자란?
  - 조건 ? 참일 때 : 거짓일 때

- 🧠 기억할 point
  - JSX 템플릿 안에선 **if문**은 **사용 불가**
  - 대신 **삼항 연산자, 논리연산자**로 **대체**

```jsx
function App() {
  const isLoggedIn = true;
  const hasMessage = false;
  const userName = "윤석";

  if (!userName) {
    return <div>사용자 이름이 없습니다.</div>
  }

  return (
    <div>
      {/* 1. 삼항 연산자 */}
      <h2>
        {isLoggedIn ? `${userName}님 환영합니다!` : "로그인이 필요"}
      </h2>
      {/* 2. && 연산자 */}
      {hasMessage && <p>📨 새로운 메시지가 도착했습니다.</p>}
      {/* 3. 여러 조건 조합 */}
      {isLoggedIn && !hasMessage && <p>메시지는 없습니다.</p>}
    </div>
  )
}
```

4.2 JSX 문법 소개

- **JSX 템플릿 문법 – 반복문**
- **특정 요소들을 반복해서 보여줘야 하는 경우**
  - **.map() 함수 사용!**
  - for문은 사용 불가
  - 조건에 맞는 요소만 표시하려면 **.filter() 함수와 조합**

- **🧠 기억할 point**
  - JSX 내부에선 for 사용 X
  - -> map()을 사용하는 것이 JSX 반복의 표준 방식

```jsx
function App() {
  const fruits = ["🍎 사과", "🍌 바나나", "🍊 오렌지"];
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
}
```

- 🍎 사과
- 🍌 바나나
- 🍊 오렌지

```jsx
function App() {
  const items = [1, 2, 3, 4];
  return (
    <ul>
      {items
        .filter(n => n % 2 === 0)
        .map(n => <li key={n}>짝수: {n}</li>)
      }
    </ul>
  );
}
```

- 짝수: 2
- 짝수: 4

4.3 Props를 통한 데이터 전달

- 리액트의 UI는 JSX라는 함수로 구성...
- 함수라면 Input과 Output이 있어야 함!

- 여기서 Output은 UI 구성 정보인 JSX이고, Input은 바로 Props로 받음
- 단, Props는 항상 위에서 아래로(부모 → 자식) 전달됨

JSX에서 사용하는 함수 컴포넌트는 원래 props 객체를 인자로 받음
그리고 보통은 props.name처럼 꺼내 씀
하지만 함수의 매개변수에서 구조분해할당 { name }을 사용하면,
props.name을 직접 쓰는 대신 name을 바로 쓸 수 있게 됨.

```
function Child({name}: { name: string }) {
  return <li>안녕하세요, {name}님!</li>;
}

function App() {
  const fruits = [" 사과", " 바나나", " 오렌지"];
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <Child key={index} name={fruit} />
      ))}
    </ul>
  );
}
```

- 안녕하세요, 사과님!
- 안녕하세요, 바나나님!
- 안녕하세요, 오렌지님!

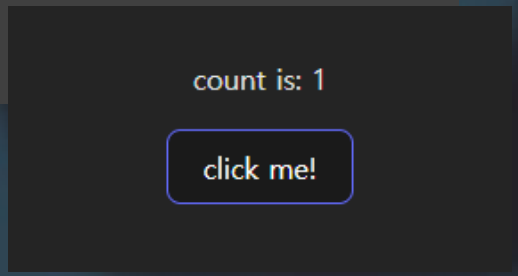
4.4 useState로 상태 관리하기

- React = "반응하다“
- UI가 상태(state)에 따라 자동으로 반응해야 함
- React는 이런 동적인 UI를 쉽게 만들 수 있도록 도와주는 라이브러리

- 버튼을 클릭하면 숫자가 증가해야 할 때,
- 그 숫자라는 “변하는 값”을 상태(state)라고 부름

- 상태가 바뀌면 React는 자동으로 해당 UI 부분을 다시 그려줍니다.
- 이를 위해 사용하는 것이 바로 useState()
  - const [count, setCount] = useState(0);
  - const [변수명, set변수명] = useState(초깃값);

```jsx
function App() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>count is: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        click me!
      </button>
    </div>
  );
}
```



4.4 useState로 상태 관리하기

- 리액트는 **성능을 위해 여러 상태 업데이트를 한꺼번에 처리(배치 처리)**합니다.
- 그래서 setState를 여러 번 써도, 이전 값이 반영되지 않으면 이상한 결과가 나올 수 있음
- **이런 경우에는 반드시 함수형 업데이트를 사용해야 함**
- **setCount(count + 1) 말고 setCount(prev => prev + 1)처럼 요!**

- **특히 사용자가 버튼을 빠르게 여러 번 누르거나, 아이템을 빠르게 담는 쇼핑몰 같은 곳에서는**
- **이걸 제대로 처리하지 않으면 클릭한 걸 놓치게 됩니다.**

```
function App() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>count is: {count}</p>
      <button onClick={() => {
        setCount(count + 1);
        setCount(count + 1);
        setCount(count + 1);
      }}>
        click me!
      </button>
    </div>
  );
}
```

```
function App() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>count is: {count}</p>
      <button onClick={() => {
        setCount((prev) => (prev + 1));
        setCount(prev => prev + 1);
        setCount(prev => prev + 1);
      }}>
        click me!
      </button>
    </div>
  );
}
```

4.5 이벤트 처리: onClick, onChange 등

- HTML에서 <button>, <input> 같은 태그는 시각적인 역할은 있지만, 기능은 자동으로 연결되지 않음.
- 실제 동작(클릭, 입력 등)을 처리하려면 JavaScript 의 .addEventListner() 메서드를 사용해 이벤트를 감지하고, 그에 맞는 로직을 수동으로 작성해야 함.

- 리액트는 DOM요소에 직접 .addEventListner()를 붙이지 않고, JSX 문법을 활용해 속성 형태로 이벤트 리스너를 등록
  - Ex) 버튼에 onClick, 입력창에 onChange, 폼에 onSubmit
- 버튼뿐만 아니라 div 등에도 가능하지만, 권장하지 않음

```
function App() {
  return (
    <div>
      <button onClick={() => {
        console.log('Event handler called');
      }}>
        Click me
      </button>
      <input
        type='text'
        onChange={(e) => {
          console.log('Input changed:', e.target.value);
        }}
        placeholder='Type something...'
      />
    </div>
  );
}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

- https://tailwindcss.com/docs/installation/using-vite
- 요새 유행하는 CSS 라이브러리
- 클래스명으로 짧게 CSS를 적용 가능함

02 **Install Tailwind CSS**

Install `tailwindcss` and `@tailwindcss/vite` via npm.

Terminal

```
npm install tailwindcss @tailwindcss/vite
```

03 **Configure the Vite plugin**

Add the `@tailwindcss/vite` plugin to your Vite configuration.

vite.config.ts

```
import { defineConfig } from 'vite'
import tailwindcss from '@tailwindcss/vite'

export default defineConfig({
  plugins: [
    tailwindcss(),
  ],
})
```

04 **Import Tailwind CSS**

Add an `@import` to your CSS file that imports Tailwind CSS.

CSS

```
@import "tailwindcss";
```

05 **Start your build process**

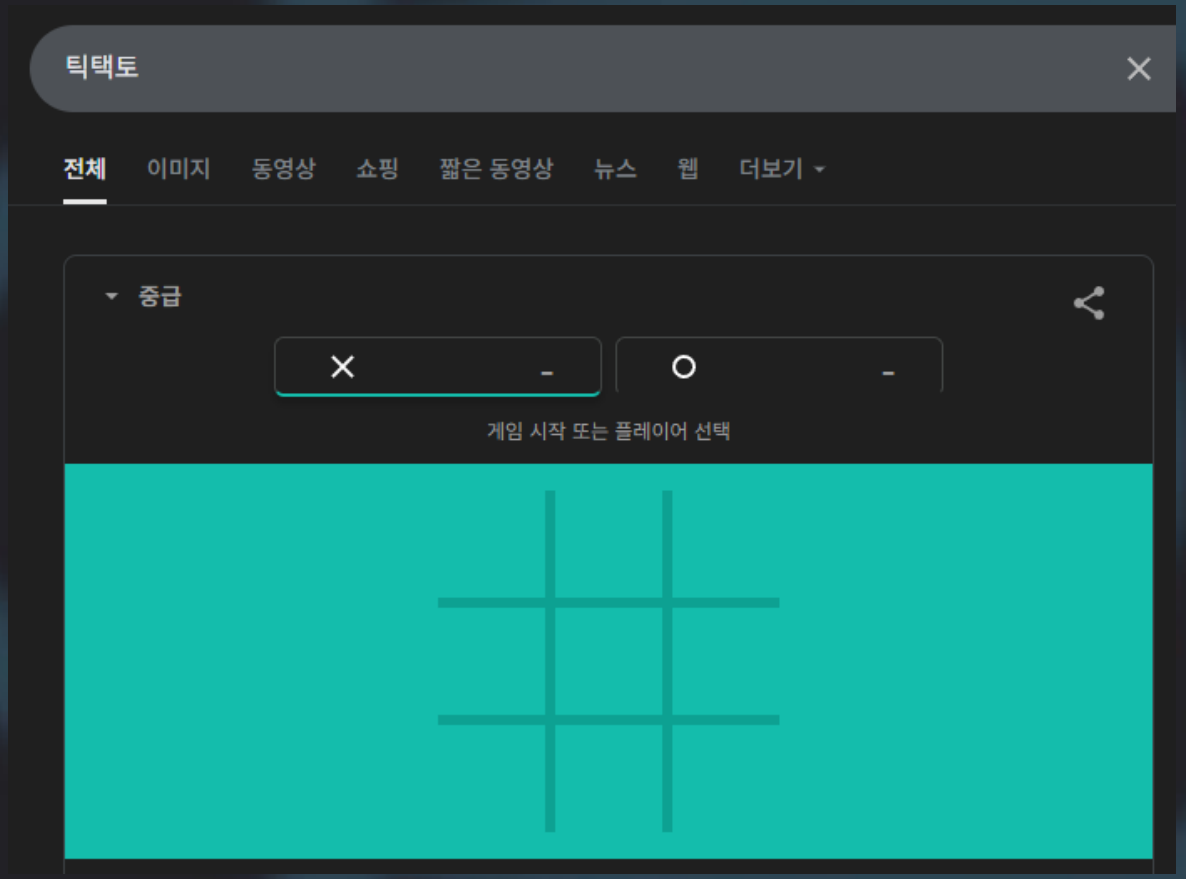
Run your build process with `npm run dev` or whatever command is configured in your `package.json` file.

Terminal

```
npm run dev
```

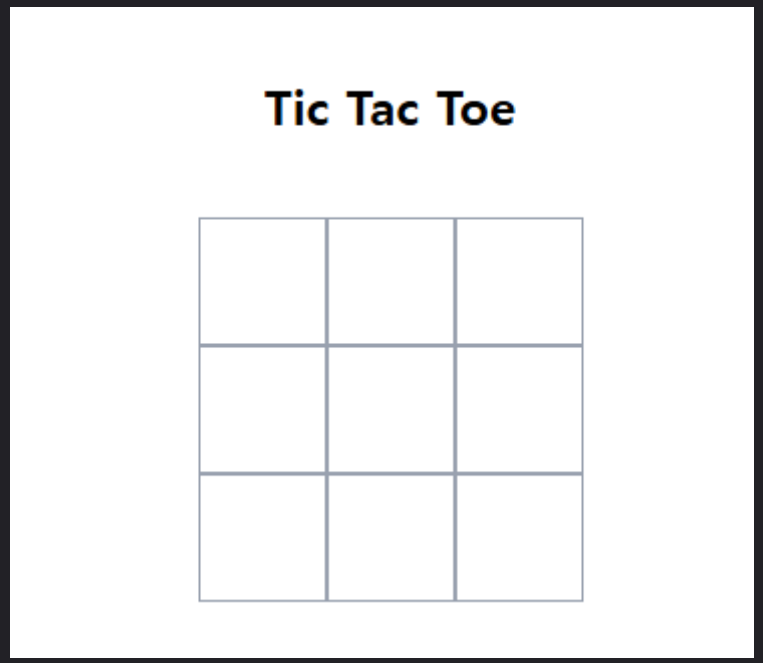
4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

- 틱택토 게임에 어떤 컴포넌트가 필요할까?
- 게임 판 (Board.tsx)
  - 3x3 셀을 배치하는 영역
- 게임 셀 (Cell.tsx)
  - 사용자가 클릭해서 X 또는 O를 표시하는 칸
- 스코어보드 (ScoreBoard.tsx)
  - 현재 턴 / 승리여부 / 리셋 버튼 등을 표시하는 영역



4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

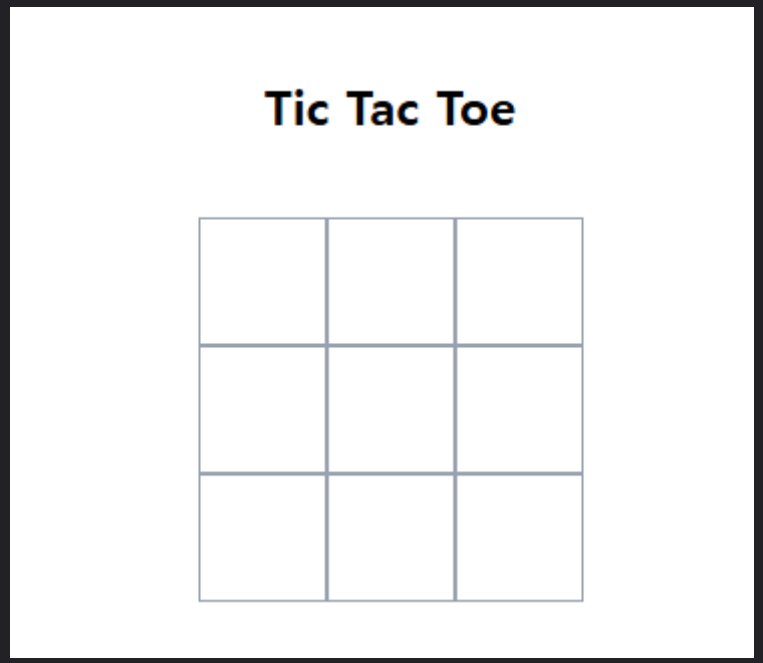
- 게임판의 역할
  - 현재 턴이 누구인가
  - 현재 어떤 판에 어떤 표식이 있는가
  - 가로 세로 대각선중 한줄로 같은 표식이 완성됐는가




```tsx
import { useState } from 'react';
export default function Board() {
  const [board, setBoard] = useState<string[][]>([
    ['', '', ''],
    ['', '', ''],
    ['', '', ''],
  ]);
  const [xIsNext, setXIsNext] = useState<boolean>(true);

  function handleClick(row: number, col: number) {
    if (board[row][col] !== '') return;   // 이미 둔 자리는 무시
    const newBoard = board.map(row => [...row]); // 깊은 복사
    newBoard[row][col] = xIsNext ? 'X' : 'O';
    setBoard(newBoard);
    setXIsNext(!xIsNext);
  }
  return (
    <div>
      <h1>Tic Tac Toe</h1>
    </div>
  )
}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

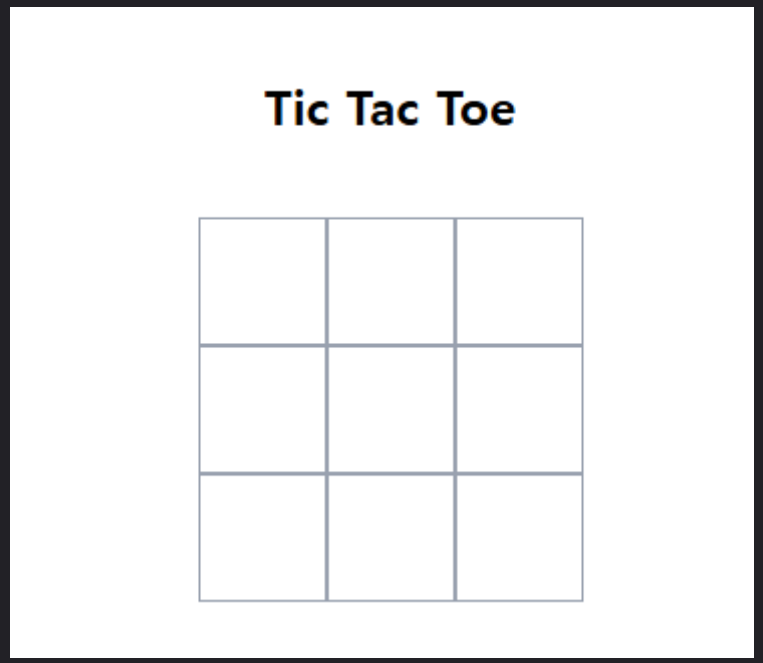
- 게임셀의 역할
  - 이 셀에 해당하는 값 표시
  - 셀 클릭시 값 바꾸기
- 셀은 자기 위치, 값을 모른다... Props로 내려주자!




```tsx
interface CellProps {
  value: string;
  row: number;
  col: number;
  handleClick: (row: number, col: number) => void;
}

export default function Cell({ value, row, col, handleClick }: CellProps) {
  const onClick = () => {
    handleClick(row, col);
  };
  return (
    <button onClick={onClick}>
      <p>{value}</p>
    </button>
  )
}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

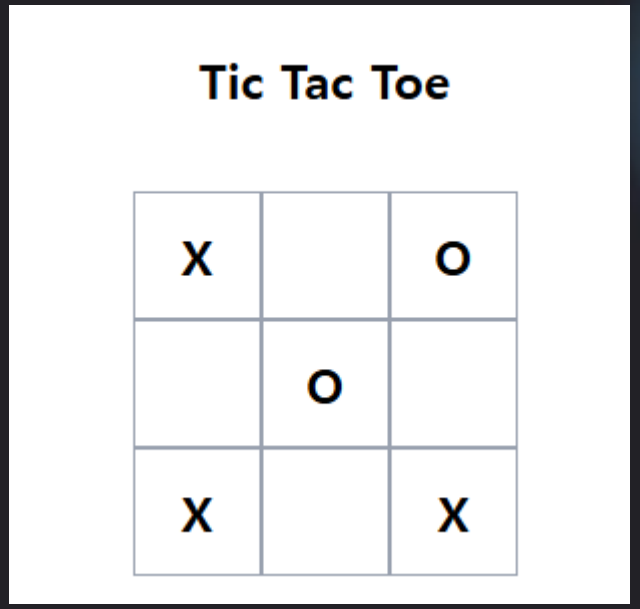
- 게임판과 게임셀 연결하기
- 앞에서 배운 map으로 2차원 배열의 판을 제작




```jsx
import { useState } from 'react';
import Cell from './Cell';
export default function Board() {
  // 이하 동일
  return (
    <div>
      <h1>Tic Tac Toe</h1>
      <div>
        {board.map((row, rowIndex) => (
          <div key={rowIndex}>
            {row.map((cell, colIndex) => (
              <Cell
                key={colIndex}
                value={cell}
                row={rowIndex}
                col={colIndex}
                handleClick={handleClick}
              />
            ))}
          </div>
        ))}
      </div>
    </div>
  )
}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

- 게임판과 게임셀 연결하기
- 앞에서 배운 map으로 2차원 배열의 판을 제작




```jsx
import { useState } from 'react';
import Cell from './Cell';
export default function Board() {
  // 이하 동일
  return (
    <div>
      <h1>Tic Tac Toe</h1>
      <div>
        {board.map((row, rowIndex) => (
          <div key={rowIndex}>
            {row.map((cell, colIndex) => (
              <Cell
                key={colIndex}
                value={cell}
                row={rowIndex}
                col={colIndex}
                handleClick={handleClick}
              />
            ))}
          </div>
        ))}
      </div>
    </div>
  )
}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

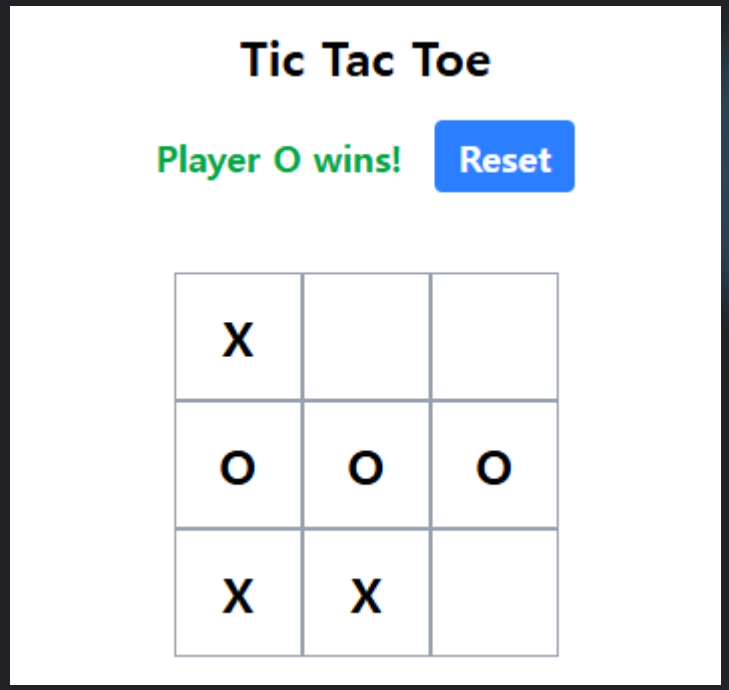
- 우승 확인하는 유틸 함수 만들기
- 보드를 입력받아 가로, 세로, 대각선 같은

- .every = 모든 요소가 조건에 맞으면 true
  - http://front2.project-study.duckdns.org/#/21

```ts
export type Player = 'X' | 'O' | '';
export type Board = Player[][];

export function checkWinner(board: Board): Player | null {
  const size = board.length;
  const players: Player[] = ['X', 'O'];

  for (const player of players) {
    for (let i = 0; i < size; i++) {
      const rowWin = board[i].every(cell => cell === player);
      const colWin = board.every(row => row[i] === player);
      if (rowWin || colWin) return player;
    }

    const diag1Win = board.every((row, idx) => row[idx] === player);
    const diag2Win = board.every((row, idx) => row[size - 1 - idx] === player);
    if (diag1Win || diag2Win) return player;
  }

  return null;
}
```

4.6 컴포넌트 구조화 실습 (

- 해당 함수를 불러오고, 우승자 상태를 추가후, 우승자가 나올 시 초기화 버튼을 조건부 렌더링하기




```tsx
import { checkWinner } from '../utils/check-winner';
import type { Player } from '../utils/check-winner';
export default function Board() {
  // 이하 동일
  const [winner, setWinner] = useState<Player | null>(null);
  function handleClick(row: number, col: number) {
    // 이하 동일..
    const result = checkWinner(newBoard);
    if (result) {
      setWinner(result);
    }
  }
  function resetGame() {
    setBoard([
      ['', '', ''],
      ['', '', ''],
      ['', '', ''],
    ]);
    setXIsNext(true);
    setWinner(null);
  }
  return (
    <div>
      <h1>Tic Tac Toe</h1>
      {winner && (
        <div>
          Player {winner} wins!
          <button onClick={resetGame}>Reset</button>
        </div>
      )}
```

4.6 컴포넌트 구조화 실습 (틱택토 게임 만들기)

- 좀더 발전시켜보기!
  - 결과를 localstorage에 저장하기
  - 각 판의 우승 결과를 모두 표시하는 점수판 만들기(ScoreBoard.tsx)
  - 각 턴을 명시적으로 보여주기
  - 새로운 게임 만들기 (ex. 단어 맞추기 게임)

오늘은 여기까지

- 오늘 배운 것
  - 컴포넌트 개념과 역할
  - JSX 문법 (중괄호, map, 삼항 연산자)
  - Props를 통한 데이터 전달
  - useState로 상태 관리
  - onClick, onChange 등 이벤트 처리

- 틱택토 실습 코드가 궁금하시면
  - https://github.com/karpitony/caps-react-study

- 수고 많으셨습니다!

다음 수업 공지

- 개인 사정으로 인해 8월 13일(수) 강의는 사전 녹화 영상으로 제공됩니다.
- 해당 주차에는 출석 체크가 없으니 편한 시간에 시청해 주세요!
- 8월 20일(수) 오후 3시, 다시 웹엑스로 뵙겠습니다.