

# K8S Demo Project

Organizing Frontend/Backend/Web/DB

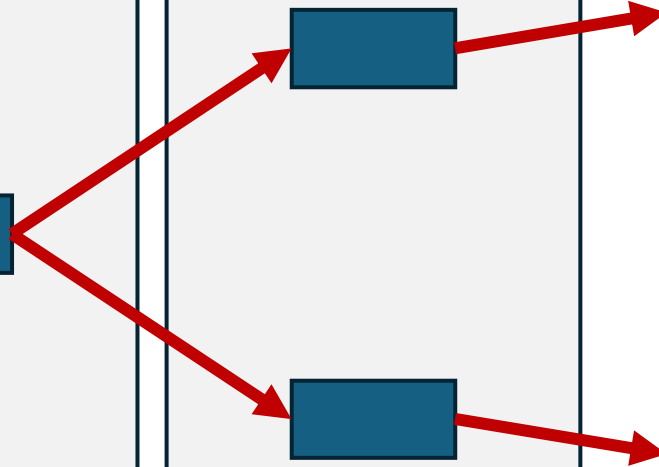
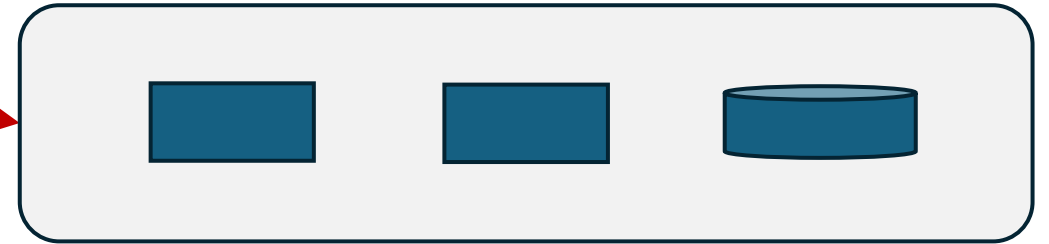
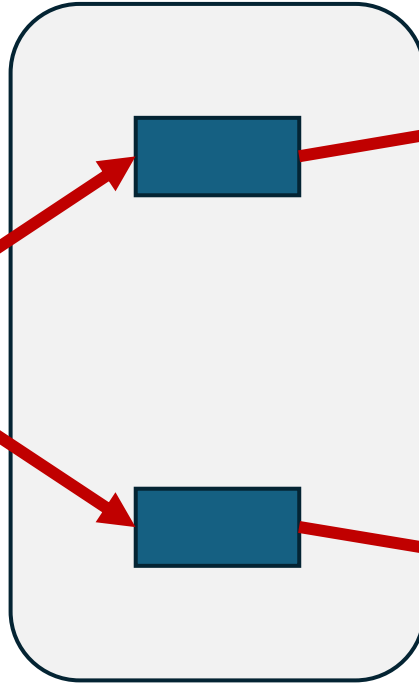
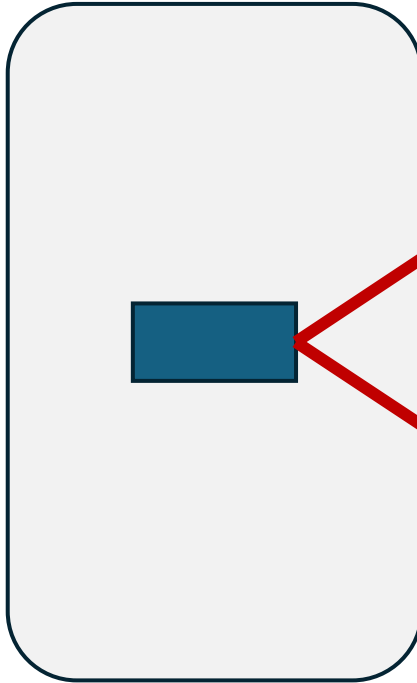
모바일시스템공학과 조민욱

adrd1820@gmail.com

# K8S

- K8S를 활용하여 아주 단순한 웹 서비스를 직접 배포해 봅시다.
- 웹 구현 자체는 너무 완벽하지 않아도 됩니다.
- 필수적으로 구현할 부분
  - 로그인을 통한 DB 참조 후 사용자 인증
  - 로그인 후 사용자 점수 Modify 기능
- 권장 기초 역량
  - 웹에 대한 최소한의 이해 (Get, Post, URL, Port, Route, Session 등)
  - Frontend, Backend, DB
  - 전반적인 Overview 개요
  - 보고 아 그렇구나... 이 정도면 오케이

REACT → CORS



# 1 - 로그인 페이지

ID

PW

LOGIN

# 1 - 로그인 페이지 - 성공

ID

PW

LOGIN

# 1 - 로그인 페이지 - 실패

로그인 실패했습니다

ID

PW

LOGIN

# 2-점수 페이지

점수는 xxxx 점입니다

SCORE

LOGIN

# Guideline

- 어떤 프레임워크를 사용해도 됩니다.
  - React, Nginx, Node.js, DB를 결합해서 사용해도 됩니다.
  - Django, DB를 결합해서 사용해도 됩니다.
- 처음 하면 매우 어려운 Challenge 과제입니다.
  - 숨 크게 들이쉬고,
  - 심호흡 하고,
  - 멘탈 잡고,
  - 들어가 봅시다.



# Django

- `python3 manage.py startapp scoreboard`
  - Google.com/shopping
  - Goolge.com/book
  - 이런 식으로 이루어진 하나의 도메인에 대한 하위 경로(세부주소)
- Localhost/scoreboard
- Localhost/test
- Localhost/demo
- 장고에서는 하위 경로 자체가 하나의 ‘앱’으로 존재
  - 디렉토리 상으로 프로젝트 명과 동일한 파일
  - 하위 경로 앱 - 1
  - 하위 경로 앱 - ...
  - Manage 파이썬 파일
  - 데이터베이스 파일

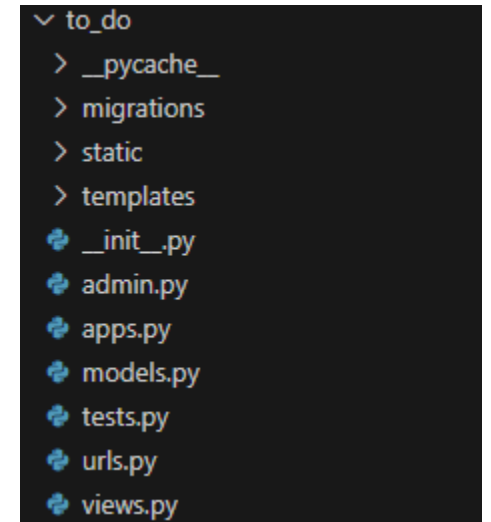
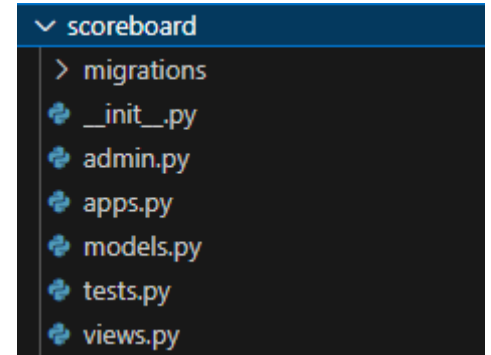
```
▼ my_demo_2 / mytest
  > mytest
  > scoreboard
  ≡ db.sqlite3
  🌀 manage.py
```

# Django

- 모델의 변경사항을 DB 스키마에 적용(by manage.py)
  - `python3 manage.py migrate` → 처음 한번
  - `python3 manage.py makemigrations scoreboard` → 앱의 변화에 따라 여러 번
- 슈퍼 유저 생성
  - `python3 manage.py createsuperuser`
  - `http://127.0.0.1:8000/admin/`

# Django

- 앱의 구성 요소
- 모델이란? 데이터 베이스 객체(사용자, 작성 글 등 어떤 것도 가능)
- `__init__.py`
- `Admin.py` → 모델 적용
- `Apps.py`
- `Models.py` → 모델 생성
- `Tests.py`
- **`Views.py`** → 내부 backend 로직, HTML을 렌더링해서 반환
- **`Urls.py`** → 따로 생성해줘야 하는 요소, 웹 접속 경로 관리



# Django

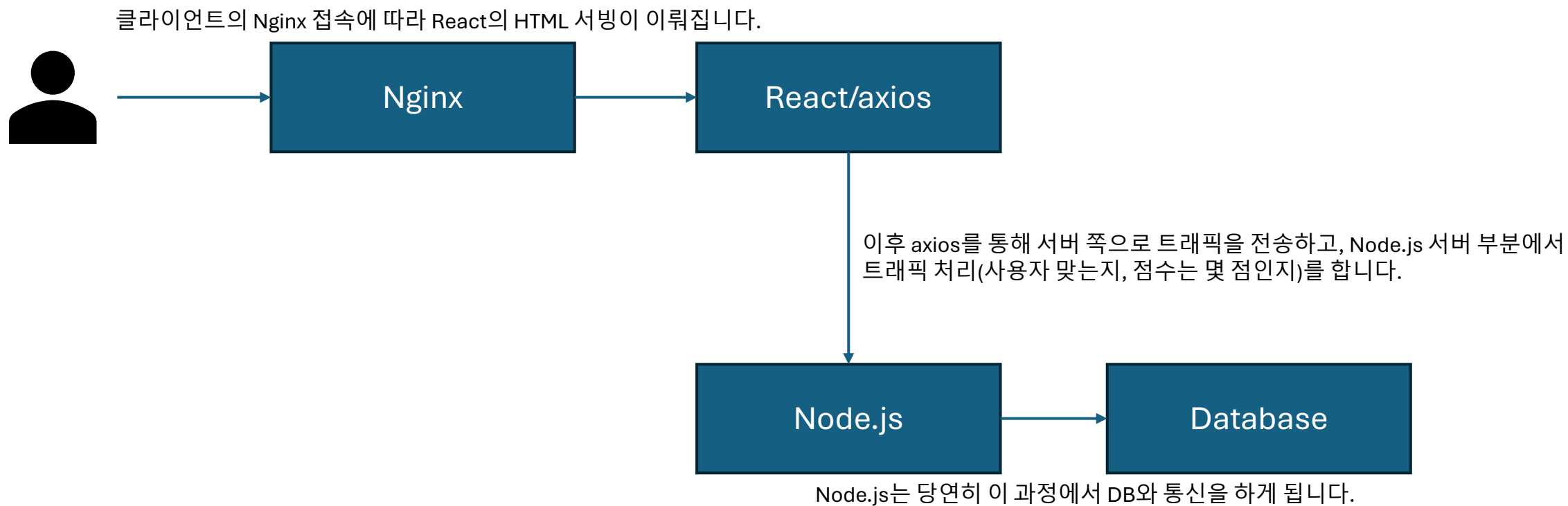
- 내 프로젝트명을 그대로 가진 또 다른 폴더
- 세부 앱에 대한 정보를 함유한 나의 서비스 근본
- `__init__.py`
- `Asgi.py`
- **Settings.py** → static 경로 등 여러 환경 설정
- **Urls.py** → 전체 앱 URL 통합 관리
- `Wsgi.py`

# Django

- My-demo 폴더 생성 및 이동
- `sudo apt install python3-pip`
- `pip install Django`
- `sudo apt install python3-Django`
  
- `django-admin startproject mytest`
- `cd mytest`
- `python3 manage.py runserver`

# React

## React 데모 프로젝트 개요



# React

- `curl -fsSL https://deb.nodesource.com/setup\_22.x -o nodesource_setup.sh`
- `sudo -E bash nodesource_setup.sh`
- `sudo apt-get install -y nodejs`
- `node -v`
- `npm -v`
  
- `cd <프로젝트 루트 명>`
- `npx create-react-app frontend`
- `cd frontend`
- `npm install http-proxy-middleware`
- `npm install react-router-dom`
- `npm start`

# React

- MY-APP/public → 이미지, HTML 등 정적 파일
- MY-APP/src → 실제 작업 파일
  - Index.html
  - Index.js
  - App.js
- Index.js 호출 → App.js를 통해 실제 보여지는 화면 렌더링 → Index.html 전달
- npm install axios
  - Connection between React and Server
  - API를 통해 사용자 정보를 전송



## App.js

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

## Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

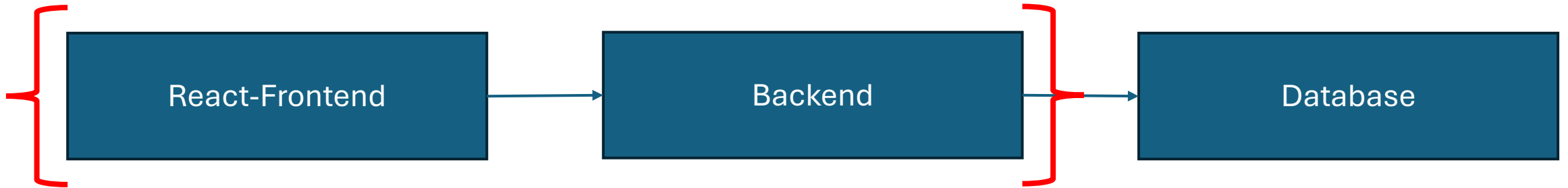
## Index.html

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
```

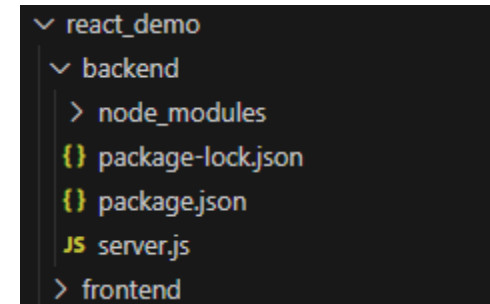
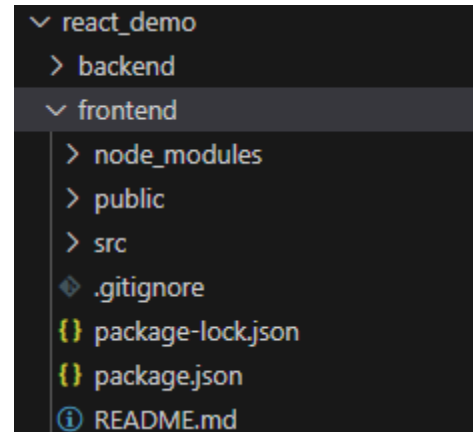
# React



- How?
  - Axios, Fetch → 서버 측과 트래픽 교환
  - Router-dom → URL 경로 관리
  - Node.js → 서버 측 언어
- 일단 이 세가지를 활용한 웹페이지를 로컬에서 테스트해봅시다.
- 물론 원래는 Node.js에서 DB 접근을 해야 합니다.

# React

- cd backend
- npm init -y
- npm install express body-parser
- npm install cors
- Backend 편집
  - server.js (node server.js 명령어로 서버 실행)
- Frontend 편집
  - Api.jsx
  - App.js
  - App.css
- CSS를 제외하면 사실상 세개의 파일!



Frontend

Api.jsx

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000', // Express 서버의 주소
  timeout: 10000, // 요청 제한 시간 설정 (옵션)
  headers: {
    'Content-Type': 'application/json',
  },
});

export default api;
```

Frontend

App.js

```
import './App.css';
import React, { useState } from 'react';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import api from './Api';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/score" element={<Score />} />
      </Routes>
    </BrowserRouter>
  );
}

function Home() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();
    await new Promise((resolve) => setTimeout(resolve, 1000));

    try {
      const response = await api.post('/check_login', {
        username: username,
        password: password
      });

      if (response.status === 200) {
        sessionStorage.setItem('username', response.data.username);
        sessionStorage.setItem('score', response.data.score);
        alert('로그인 성공했습니다.');
```

Backend

Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors'); // cors 미들웨어 추가

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(bodyParser.json());
app.use(cors()); // 모든 출처에서의 요청을 허용

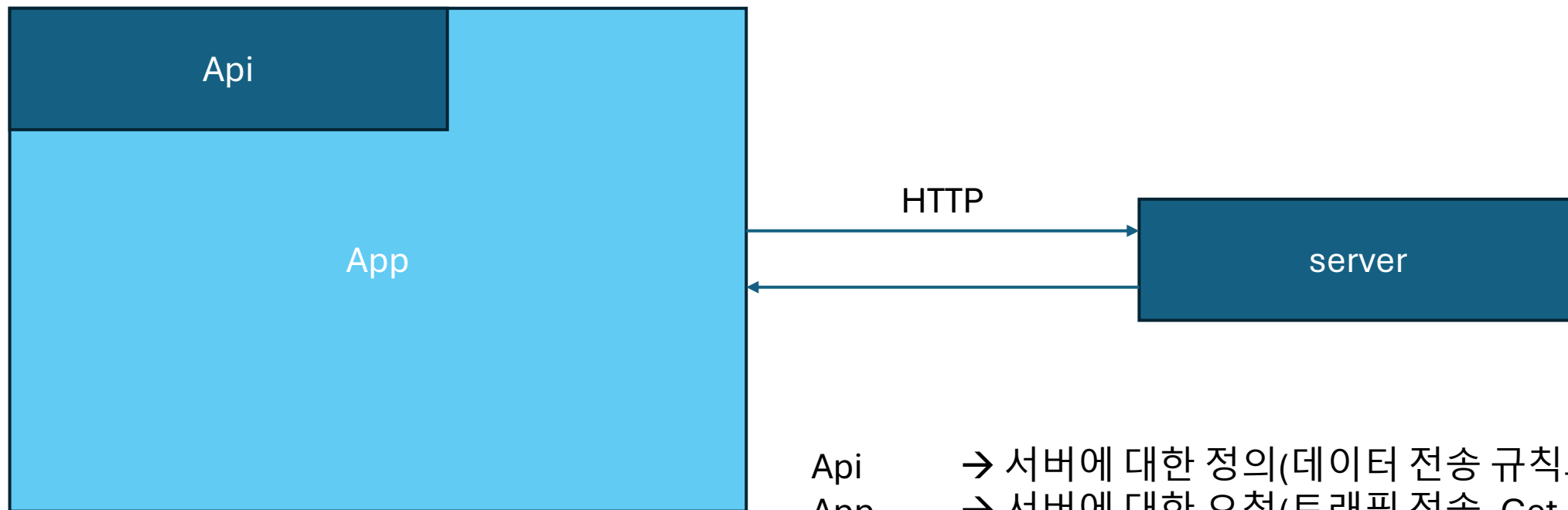
app.get('/', (req, res) => {
  res.send('Hello, World!');
});

// Simulated user data (in a real app, use a database)
let users = [
  { id: 1, username: 'user1', password: 'password1', score: 100 },
  { id: 2, username: 'user2', password: 'password2', score: 150 },
];

// POST endpoint to handle login check
app.post('/check_login', (req, res) => {
  const { username, password } = req.body;

  // Find user by username and password
  const user = users.find(u => u.username === username && u.password === password);

  if (user) {
    // If user found, send user data back to client
    res.status(200).json({ username: user.username, score: user.score });
    console.error('로그인');
  } else {
```

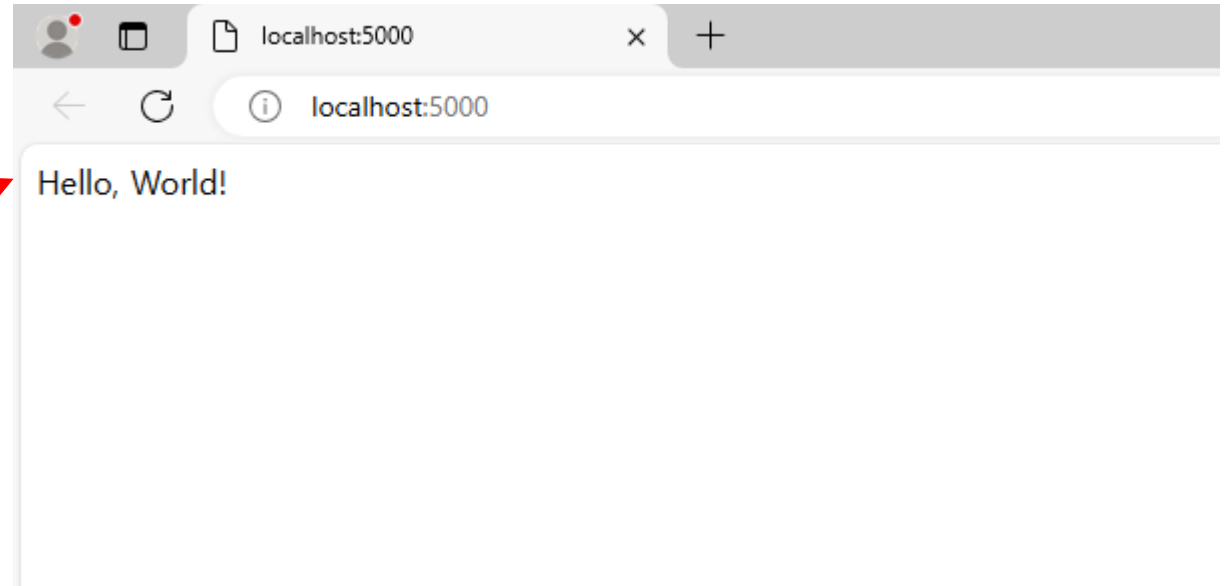


Api      → 서버에 대한 정의(데이터 전송 규칙과 방법 등)  
App      → 서버에 대한 요청(트래픽 전송, Get, Post)  
Server   → 요청에 대한 Error, 200, 404 등의 Response

# React

```
minuk@minukubuntu:~/react_demo/backend$ node server.js  
Server is running on http://localhost:5000
```

```
app.get('/', (req, res) => {  
  res.send('Hello, World!');  
});
```





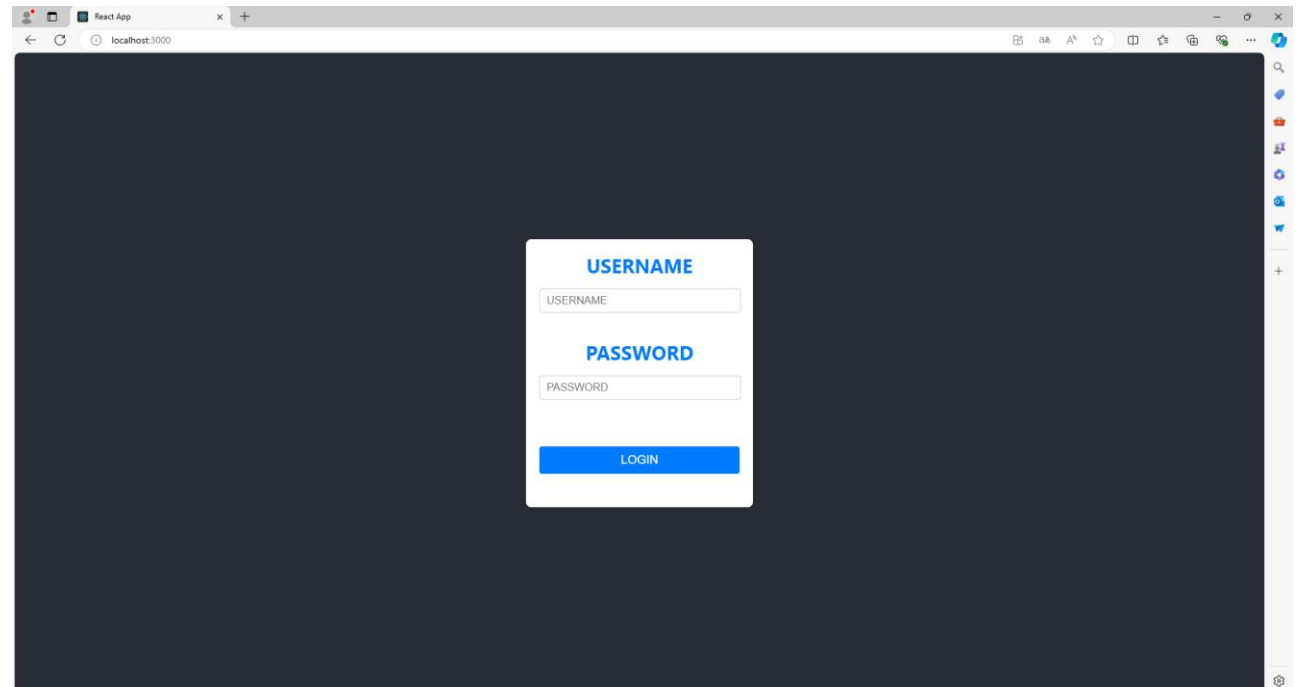
# React

```
minuk@minukubuntu:~/react_demo/frontend$ npm start
```

```
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/score" element={<Score />} />  
      </Routes>  
    </BrowserRouter>  
  );  
}
```

```
function Home()
```

```
  return (  
    <div className="App">  
      <header className="App-header">  
        <form className="login" onSubmit={handleLogin}>  
          <label htmlFor="username">USERNAME</label>  
          <input type="text" placeholder="USERNAME" onChange={(e) => setUsername(e.target.value)} />  
          <label htmlFor="password">PASSWORD</label>  
          <input type="password" placeholder="PASSWORD" onChange={(e) => setPassword(e.target.value)} />  
          <button type="submit">LOGIN</button>  
        </form>  
      </header>  
    </div>  
  );  
}
```



```
minuk@minukubuntu:~/react_demo/backend$ node server.js  
Server is running on http://localhost:5000  
Invalid username or password
```

localhost:3000의 메시지

로그인 오류가 발생했습니다.

확인

USERNAME

1234

PASSWORD

....

LOGIN

localhost:3000의 메시지

로그인 성공했습니다.

확인

USERNAME

PASSWORD

LOGIN

user1

SCORE

APPLY

```
minuk@minukubuntu:~/react_demo/backend$ node server.js
Server is running on http://localhost:5000
Invalid username or password
Invalid username or password
로그인
```

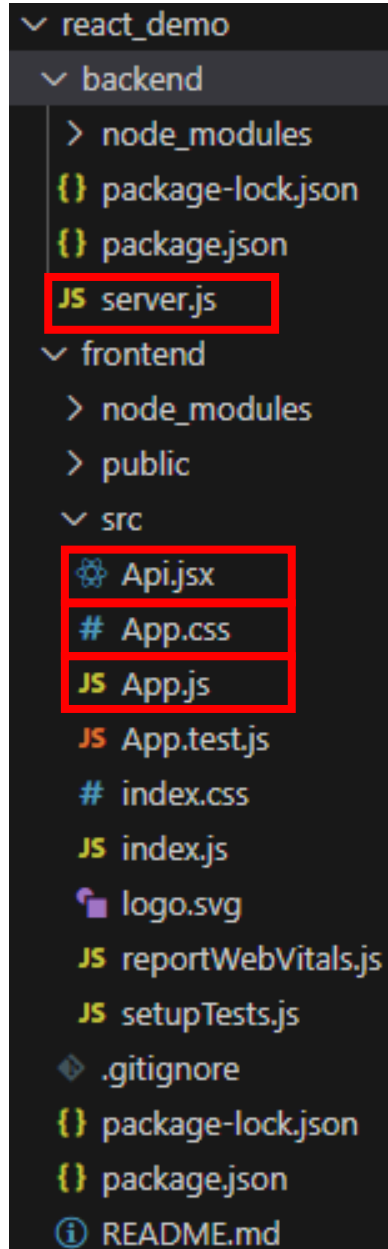
# React

---

지금까지?

4개 파일만 생성 및 편집...

Backend/Frontend에 맞춰서 Dockerfile!



```
react_demo
├── backend
│   ├── node_modules
│   ├── package-lock.json
│   ├── package.json
│   └── server.js
├── frontend
│   ├── node_modules
│   ├── public
│   └── src
│       ├── Api.jsx
│       ├── App.css
│       ├── App.js
│       ├── App.test.js
│       ├── index.css
│       ├── index.js
│       ├── logo.svg
│       ├── reportWebVitals.js
│       ├── setupTests.js
│       ├── .gitignore
│       ├── package-lock.json
│       ├── package.json
│       └── README.md
```