프론트트에서 서버에 데이터 요청하기

- 원본강의자료 : https://www.youtube.com/watch?v=d6suykcsNeY&list=PLxKtM_ITsDyuRAFbJ0vNuu-h77ENEhk7G&index=14

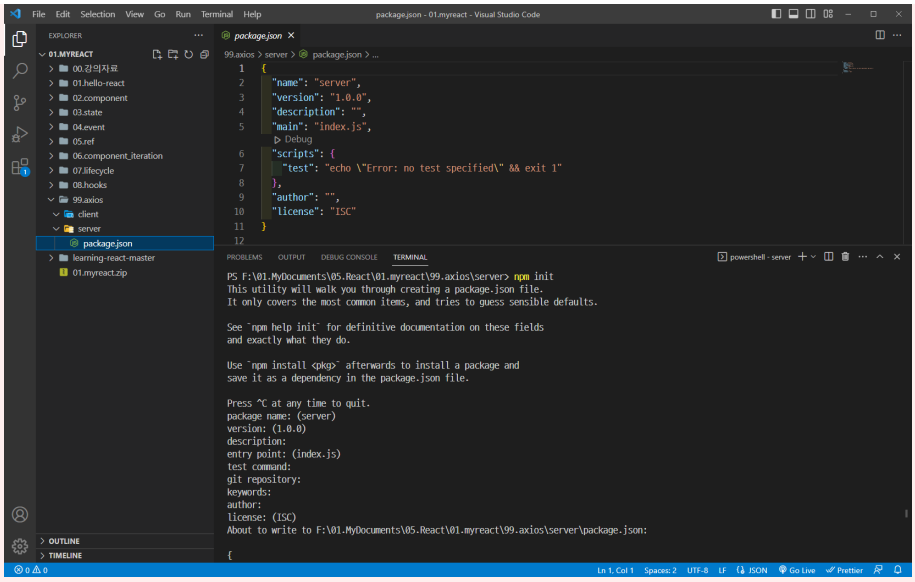
1. 작업폴더 생성 및 환경설정하기

- 99.axios
  - client
  - server

2. server

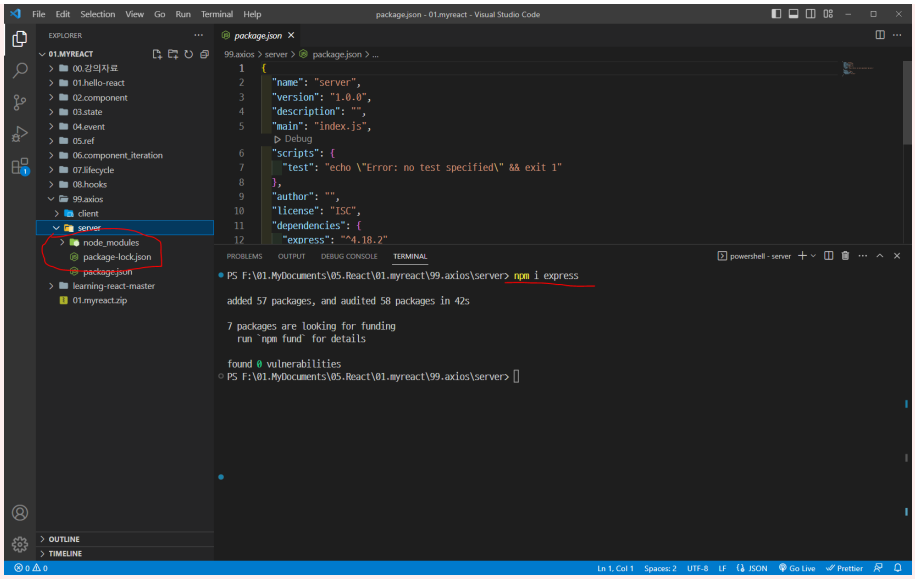
- server폴더에서 작업

1. `npm init` -> package.json 생성



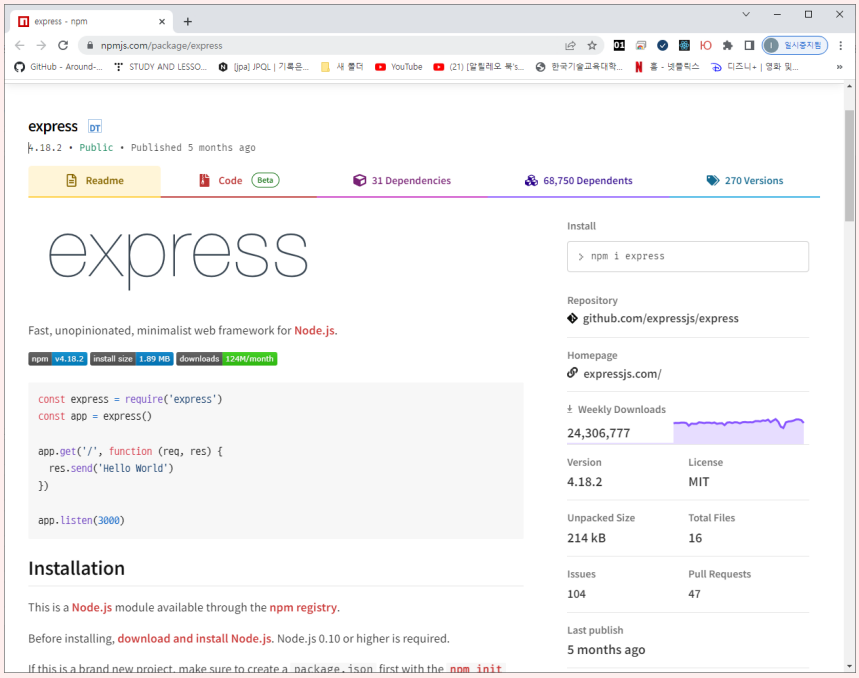
- 참고사이트 : https://docs.npmjs.com/cli/v8/commands/npm-init / https://carrotweb.tistory.com/107

2. `npm i express`



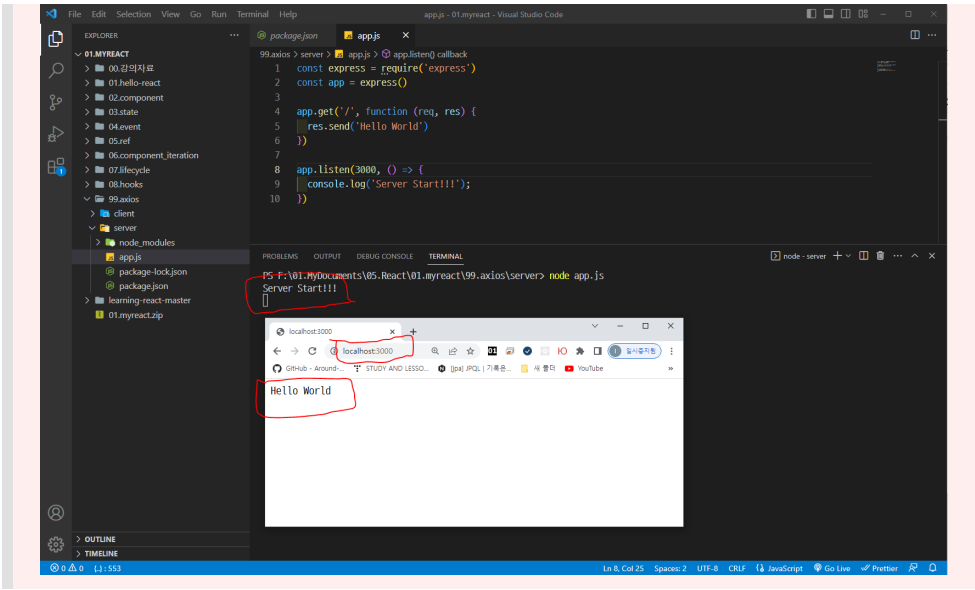
3. app.js 작성

- https://www.npmjs.com/package/express 에서 sample code 복사 후 정상실행을 확인하기 위해 `app.listen` 수정



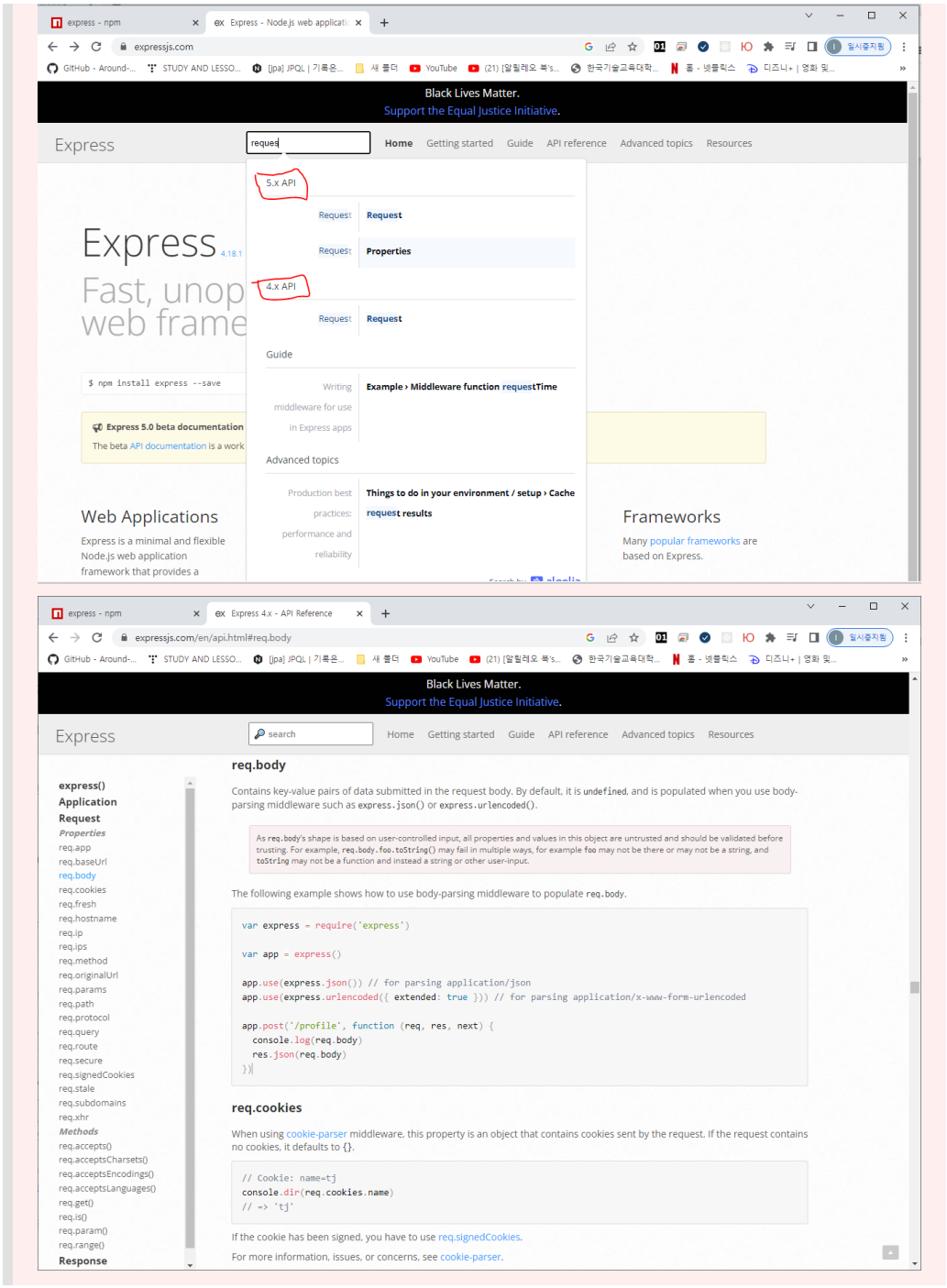

```
server\app.js
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000, () => {
  console.log('Server Start!!!');
})
```

1. `node app.js` : 서버 실행!!

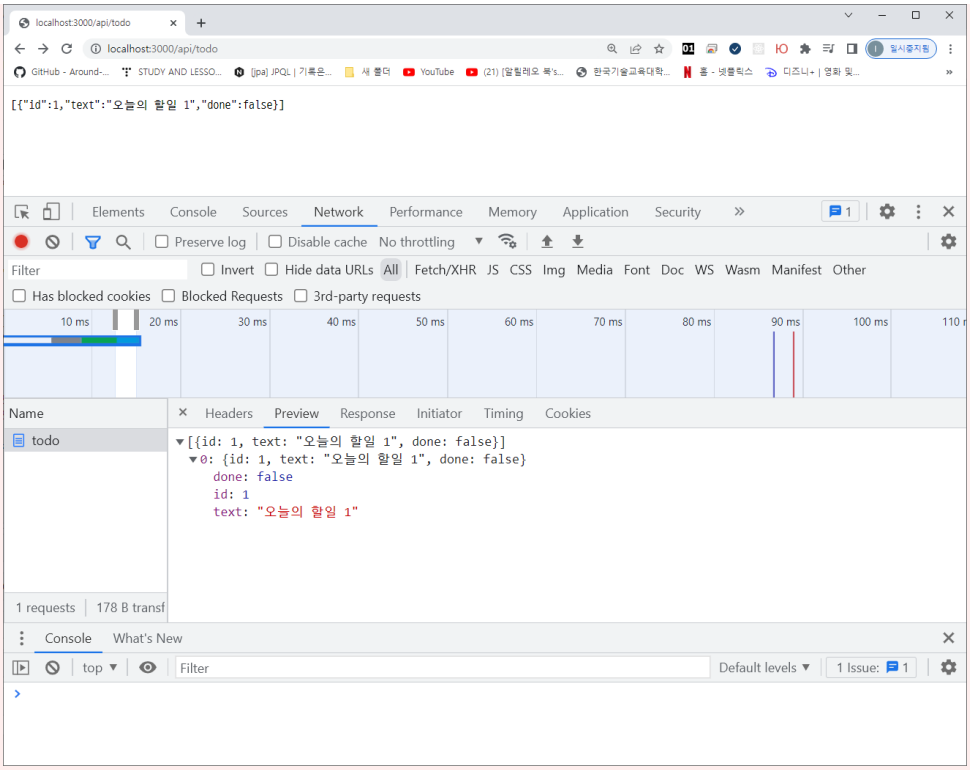


1. app.js : crud 로직 작성(1)

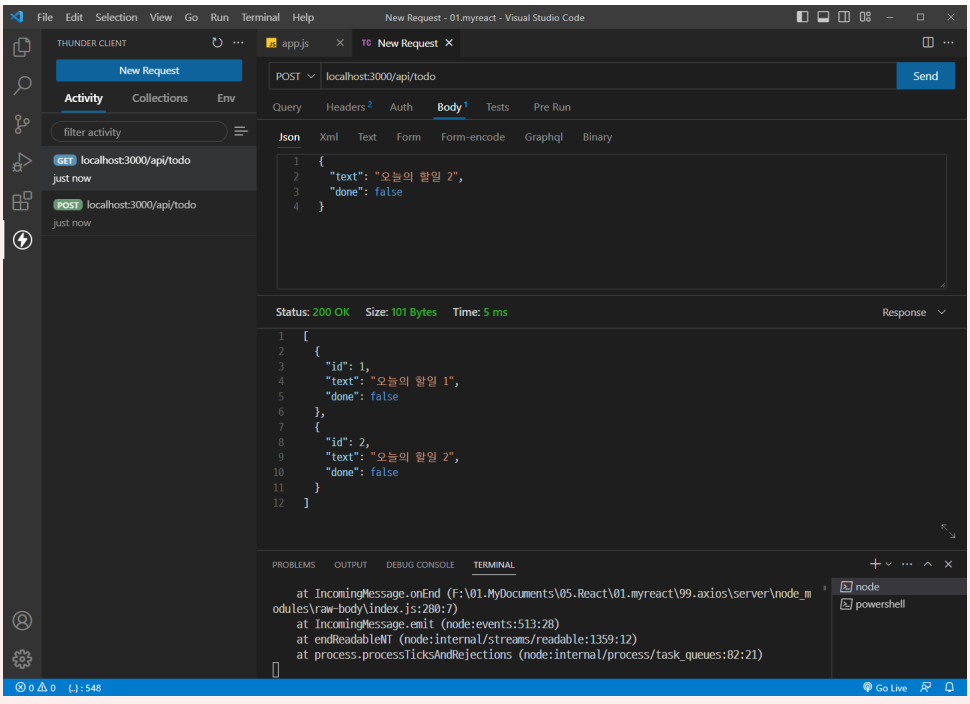
```
server\app.js
const express = require('express');
const app = express();

// expess(4.18.x기준) 에서 req.body의 데이터를 가져오려면 boaty
parser가 필요
// 샘플코드 : https://expressjs.com/en/api.html#req.body
app.use(express.json()); // for parsing application/json
app.use(express.urlencoded({ extended: true })); // for parshing

let id = 2;
const todoList = [{
  id: 1,
  text: "오늘의 할일 1",
  done: false
}];

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

app.get('/api/todo', (req, res) => {
  res.json(todoList);
});

app.post('/api/todo', (req, res) => {
  const {text, done } = req.body;
  todoList.push({
    id: id++,
    text,
    done,
  });
  return res.send("post success!!")
});

app.listen(3000, () => {
  console.log('Server Start!!!');
})
```



1. app.js 저장후 서버 재시작
   - `node app.js` : 서버 실행!!
   - web browser : `http://localhost:3000/api/todo` : get 요청



- `Rest API test` (postman 대신 Thunder Client V2.5.0 사용)
  - vscode plug in : `Thunder Client` 플러그인 설치
  - get and post 요청



server에서 정상적으로 실행 완료되었으면 이제 client에서 작업을 작성!!!

3. client

1. react app 생성 : client 폴더에서 실행

```
npx create-react-app .
```

1. 생성후 필요 없는 파일 삭제

- App.css
- App.test.js
- index.css
- logo.svg

2. index.js 와 App.js에 삭제된 파일 import문 삭제

3. App.js 작성

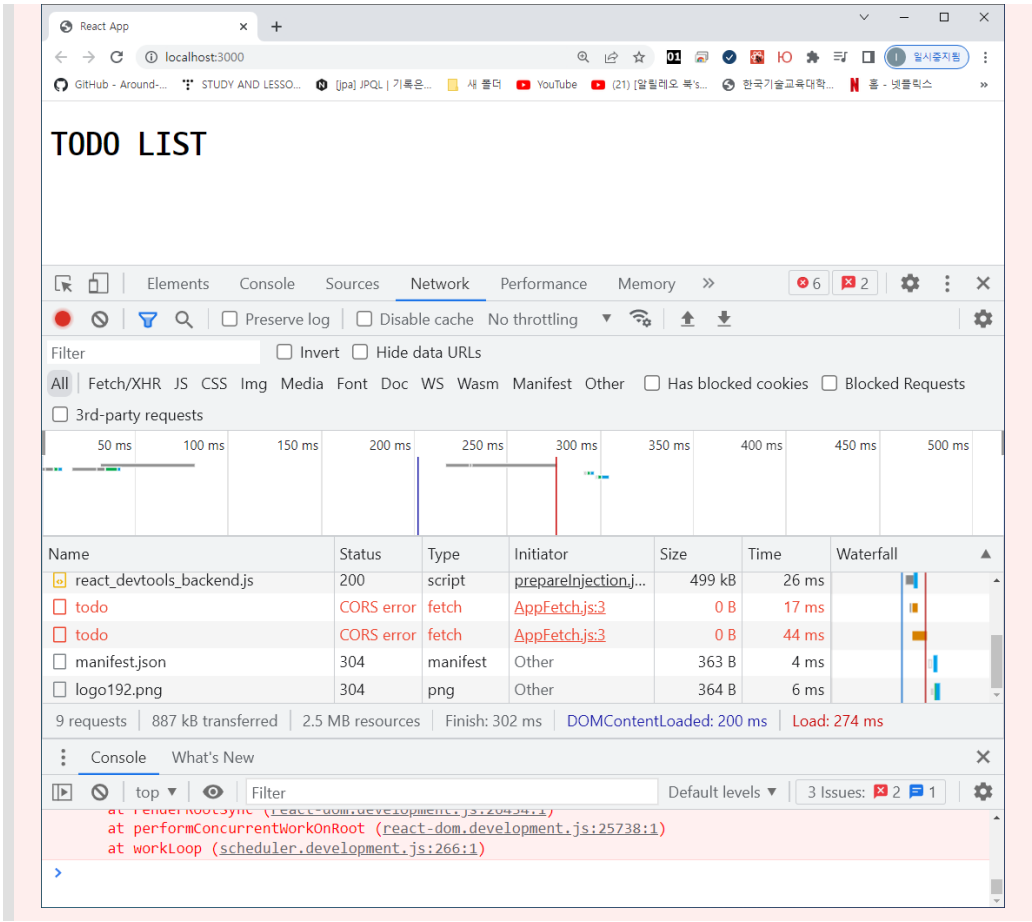
- 서버와 Rest API통신을 하기 위한 라이브러리
  1. fetch : 기본 제공 라이브러리
  2. axios : 별도 설치 plug in

3.1 fetch

- fetch 사용법
  - https://developer.mozilla.org/ko/docs/Web/API/Fetch_API/Using_Fetch

1. 서버연동하기

```
index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import AppFetch from './AppFetch';
import reportWebVitals from './reportWebVitals';

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

reportWebVitals();

AppFetch.js
function AppFetch() {

  fetch('http://localhost:4000/api/todo')
    .then((response) => response.json())
    .then((data) => console.log(data));

  return (
    <div className="AppFetch">
      <h1>TODO LIST</h1>
    </div>
  );
}

export default AppFetch;
```



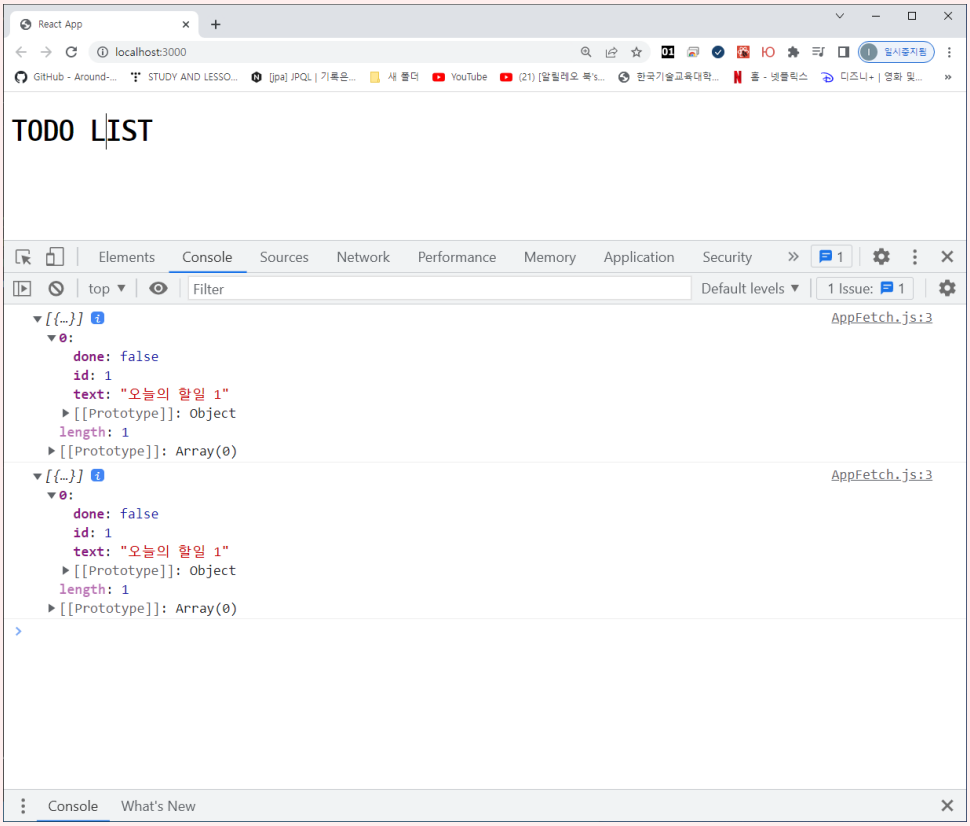
CORS(Cross Origin Resource Sharing) 에러

- CORS에러란? 예를 들어 naver.com에서 google.com 서버에 있는 자료를 제한 없이 접근할 수 있다면 많은 문제가 야기될 것이다.
- 통상의 경우 외부에서 Origin이 아닌 요청을 근본적으로 막아 놓아 외부에서 접근하지 못하도록 한다.
- client(localhost:3000)에서 servet(localhost:4000)에 접근한다는 것은 `Origin이 다르다` 는 것이다.
- Origin이 다를 경우에 서버에 접근이 가능하도록 cors에러를 허용해야 한다.
- 해결방법 : npm 홈페이지에서 cors를 검색 : https://www.npmjs.com/package/cors
  - `server(폴더)에 설치`
  - 라이브러리 설치 : `npm i cors` or `yarn add cors`
  - 설치 후 servar/app.js 코드추가(CORS정책을 풀어주기)
    - cors()함수에 매개변수를 지정하지 않으면 아무나 서버에 접근할 수 있다는 의미

```
server\app.js
const express = require('express');
const app = express();

const cors = require('cors');
app.use(cors());

--- 이하 생략 ---
```

  - 서버 재시작 : `node app.js`
- 참고 : https://inpa.tistory.com/entry/WEB-📚-CORS-💯-정리-해결-방법-👏

CORS에러 해결결과

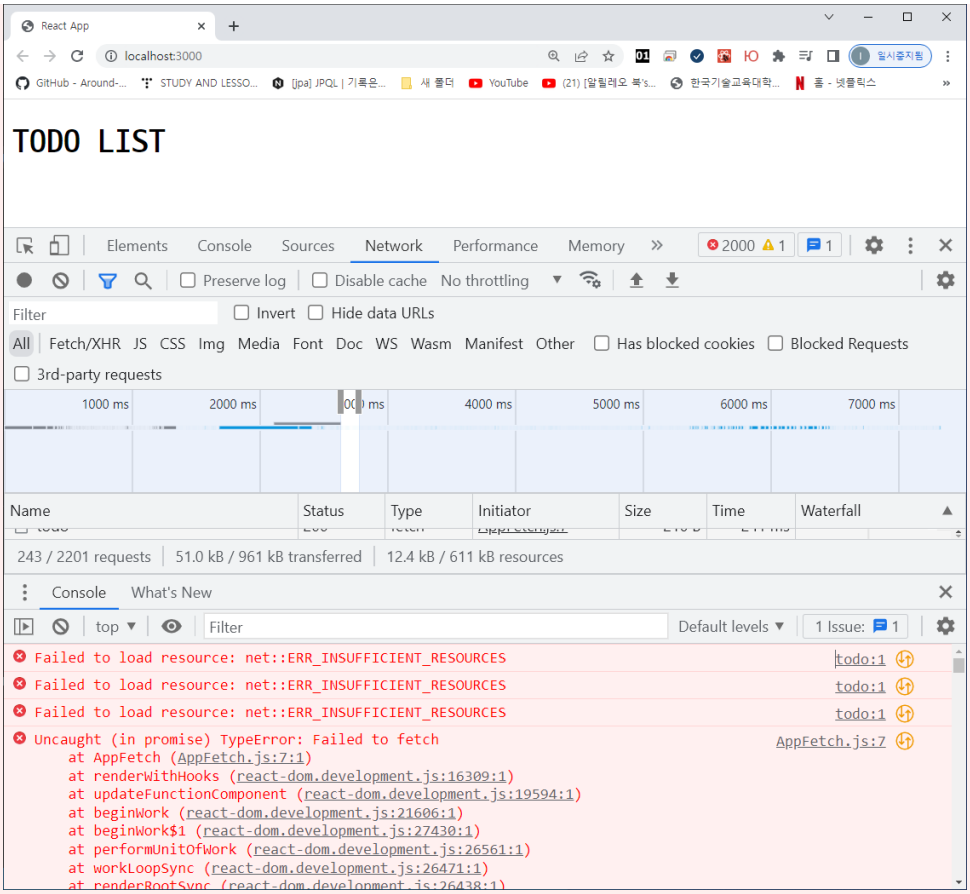


1. 클라이언트에 서버 자료 출력하기

AppFetch.js

```
import { useState } from 'react';

function AppFetch() {

  const [todoList, setTodoList] = useState(null);

  fetch('http://localhost:4000/api/todo')
    .then((response) => response.json())
    .then((data) => setTodoList(data));

  return (
    <div className="AppFetch">
      <h1>TODO LIST</h1>
    </div>
  );
}

export default AppFetch;
```

무한 Loop 에러



1. 서버에 신규자료 등록

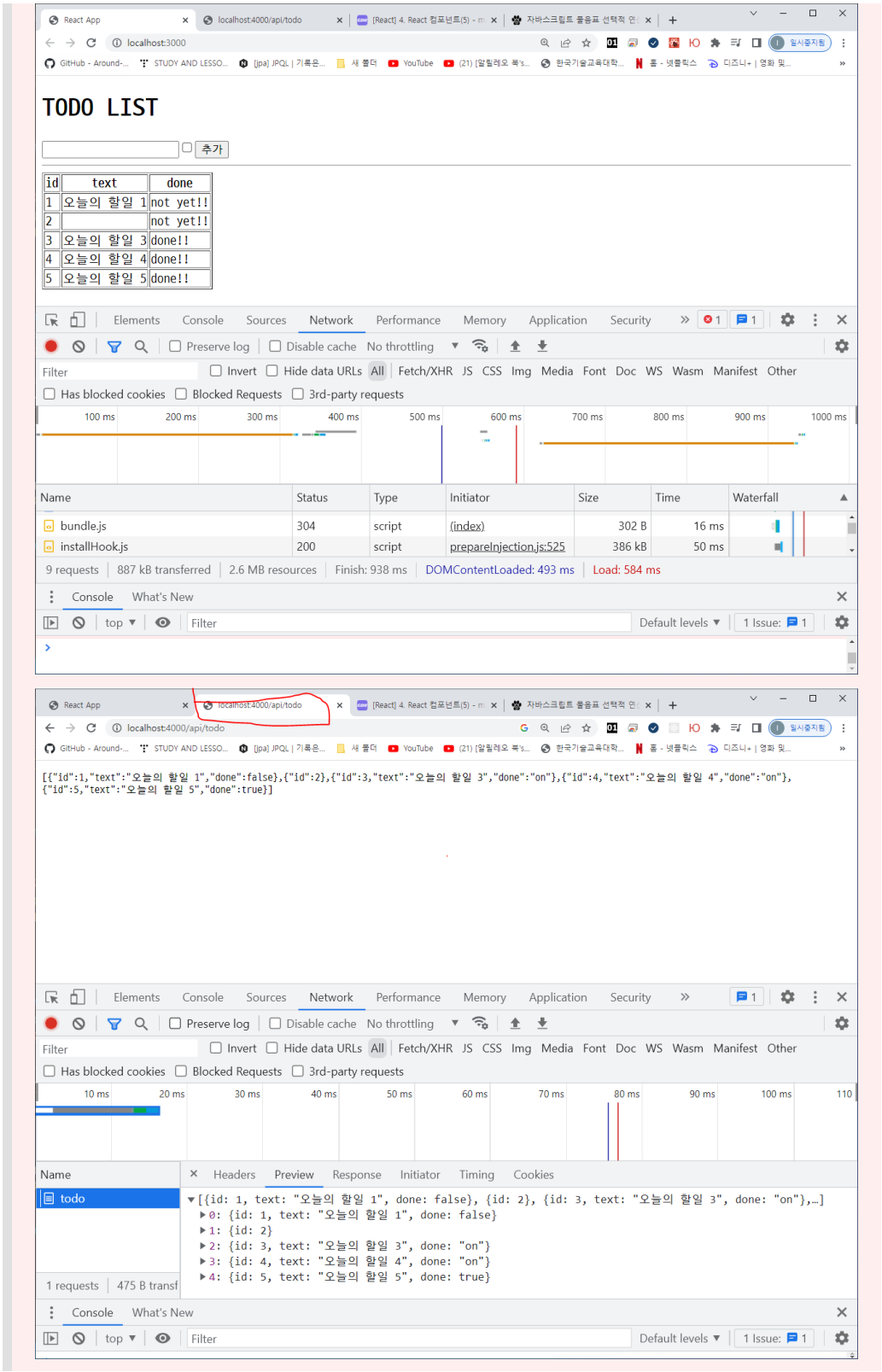
- 무한 Loop 해결
- 값이 변경될 때 랜더링되기 때문에 랜더링을 초기화 하기 위해 `userEffec hook을 사용`
- 데이터를 서버에 추가(post)
- 서버에 추가가 되지만 클라이언트에 결과가 출력되지 않는다.
- 처음에 todoList에 null값이 들어 어면 전달자료가 없기 때문에 아래와 같이 `선택적 연산자, Optional chaining 정의`

> - 객체사용시, 속성이 undefiend이거나 null일 수 있다.
> - 참조값이 undefiend나 null이면 error를 발생시키는 대신 undefiend를 return한다.
> - `todoList?.map((todo) => ( ... )`
> - 참고 : https://pusha.tistory.com/entry/자바스크립트-물음표-선택적-연산자-Optional-chaining
> - : https://joylee-developer.tistory.com/166

AppFetch.js

```
import { useState, useEffect } from 'react';

function AppFetch() {

  const [todoList, setTodoList] = useState(null);

  useEffect(() => {
    fetch('http://localhost:4000/api/todo')
      .then((response) => response.json())
```

```
      .then((data) => setTodoList(data));
  }, []);

  const onSubmitHandler = (e) => {
    e.preventDefault();
    const text = e.target.text.value;
    const done = e.target.done.checked;
    fetch('http://localhost:4000/api/todo', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        text,
        done
      })
    })
  }

  return (
    <div className="AppFetch">
      <h1>TODO LIST</h1>
      <form onSubmit={onSubmitHandler}>
        <input type="text" name="text"/>
        <input type="checkbox" name="done"/>
        <input type="submit" value="추가"/>
      </form>
      <hr/>
      <table border={"1"}>
        <tr>
          <th>id</th>
          <th>text</th>
          <th>done</th>
        </tr>
        {todoList?.map((todo) => (
          <tr key={todo.id}>
            <td>{todo.id}</td>
            <td>{todo.text}</td>
            <td>{todo.done ? 'done!!' : 'not yet!!' }</td>
          </tr>
        ))}
      </table>
    </div>
  );
}

export default AppFetch;
```



1. 신규등록자료의 클라이언트에 조회

- 서버에 신규자료가 등록이 되면 결과를 클라이언트에 조회하기 위해서 서버에 자료를 요청(get)해야 한다.
- 등록직후에 서버에 자료 요청하기 위해 `onSubmitHandler 로직을 추가`

```
const onSubmitHandler = (e) => {
  e.preventDefault();
  const text = e.target.text.value;
  const done = e.target.done.checked;
  fetch('http://localhost:4000/api/todo', {
```

```
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        text,
        done
      })
    }).then(() => {
      fetch('http://localhost:4000/api/todo')
      .then((response) => response.json())
      .then((data) => setTodoList(data));
    });
    e.target.text.value = '';
    e.target.done.checked = false;
    e.target.text.focus();
  }
```

1. 최종완성

- 결과요청 중복로직을 함수로 처리
- 추가완료후 input 박스 초기화 및 focusing
- thead, tbody 사용안할 경우 경고메시지 출력!!

AppFetch.js

```
import { useState, useEffect } from 'react';

function AppFetch() {

  const [todoList, setTodoList] = useState(null);

  const fetchData = () => {
    fetch('http://localhost:4000/api/todo')
      .then((response) => response.json())
      .then((data) => setTodoList(data));
  };

  useEffect(() => {
    fetchData();
  }, []);

  const onSubmitHandler = (e) => {
    e.preventDefault();
    const text = e.target.text.value;
    const done = e.target.done.checked;
    fetch('http://localhost:4000/api/todo', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        text,
        done
      })
    }).then(() => {
      fetchData();
    });
    e.target.text.value = '';
    e.target.done.checked = false;
    e.target.text.focus();
```

```
  }

  return (
    <div className="AppFetch">
      <h1>TODO LIST</h1>
      <form onSubmit={onSubmitHandler}>
        <input type="text" name="text"/>
        <input type="checkbox" name="done"/>
        <input type="submit" value="추가"/>
      </form>
      <hr/>
      <table border={"1"}>
        <thead>
          <tr>
            <th>id</th>
            <th>text</th>
            <th>done</th>
          </tr>
        </thead>
        <tbody>
        {todoList?.map((todo) => (
          <tr key={todo.id}>
            <td>{todo.id}</td>
            <td>{todo.text}</td>
            <td>{todo.done ? 'done!!' : 'not yet!!' }</td>
          </tr>
        ))}
        </tbody>
      </table>
    </div>
  );
}

export default AppFetch;
```

3.2 axios

1. axios 설치

   - client에 axios를 설치 : `yarn add axios`
   - axios를 설치하면 fetch보다 더 쉽게 Rest API를 구현할 수 있다.
   - 데이터를 가져오는 로직을 사용자 hook을 만들어 재사용할 수도 있다
   - 또한, 데이터 로딩처리 및 에러처리 로직도 구현할 수 있다.
   - axios보다 더 쉽게 사용할 수 있는 것이 `react-query`라는 것도 있다.
     - https://kyounghwan01.github.io/blog/React/react-query/basic/

2. index.js / AppAxios.js 작성

```
index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import AppFetch from './AppFetch';
import reportWebVitals from './reportWebVitals';
import AppAxios from './AppAxios';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {/* <AppFetch /> */}
    <AppAxios />
  </React.StrictMode>
);

reportWebVitals();
```

AppAxios.js

```
import { useState, useEffect } from 'react';
import axios from 'axios';

const SERVER_URL = 'http://localhost:4000/api/todo';

function AppAxios() {

  const [todoList, setTodoList] = useState(null);

  // async, await 미사용
  // const fetchData = () => {
  //   axios.get(SERVER_URL).then(res => setTodoList(res.data));
  // };

  // async, await 사용
  const fetchData = async () => {
    const res = await axios.get(SERVER_URL);
    setTodoList(res.data);
  };

  useEffect(() => {
    fetchData();
  }, []);

  const onSubmitHandler = async (e) => {
    e.preventDefault();
    const text = e.target.text.value;
    const done = e.target.done.checked;

    // axios는 headers에 Content-Type등의 정보를 자동을 처리해 준다.
    await axios.post(SERVER_URL, { text, done });
    fetchData();
    e.target.text.value = '';
    e.target.done.checked = false;
    e.target.text.focus();
  }

  return (
    <div className="AppFetch">
      <h1>TODO LIST</h1>
      <form onSubmit={onSubmitHandler}>
        <input type="text" name="text"/>
        <input type="checkbox" name="done"/>
        <input type="submit" value="추가"/>
      </form>
      <hr/>
```

```
        <table border={"1"}>
          <thead>
            <tr>
              <th>id</th>
              <th>text</th>
              <th>done</th>
            </tr>
          </thead>
          <tbody>
          {todoList?.map((todo) => (
            <tr key={todo.id}>
              <td>{todo.id}</td>
              <td>{todo.text}</td>
              <td>{todo.done ? 'done!!' : 'not yet!!' }</td>
            </tr>
          ))}
          </tbody>
        </table>
      </div>
    );
  }

  export default AppAxios;
```