1. axios 소개

HTTP 통신 라이브러리

- axios 이외에도 jquery ajax, fetch, superagent 등이 있음
- 이 중 axios가 최근에 가장 많이 사용됨.
- 제공 기능
  - node.js, 브라우저에서 XMLHttpRequest 객체 사용
  - Promise API를 제공함
  - 요청 데이터와 응답 데이터의 ContentType에 의한 자동 변환
  - 요청 취소(Abort) 기능 제공
  - CSRF 공격에 대한 대응을 위한 클라이언트 측 기능 지원
- IE 지원 여부 : IE8 이상
- 사용 방법
  - yarn add axios (또는) npm install axios

1

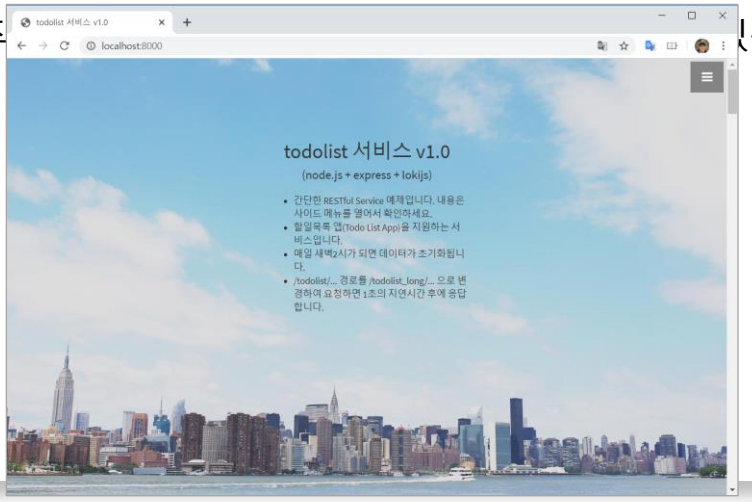
■ CDN 방식으로 사용하려면 다음과 같이 참조한다.

- <script type="text/javascript" src="https://unpkg.com/axios/dist/axios.js"></script>

2. 테스트용 API 소개(1)

TodoList API 소개

- 다음 URL 중 하나 이용
- https://todosvc.herokuapp.com
- https://todosvc.vercel.app/
- 직



음

-

2

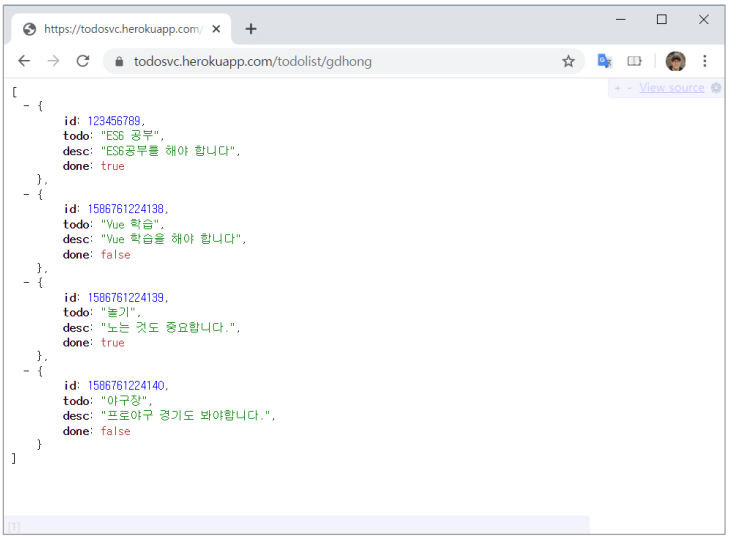
■ 구체적인 사용 방법은 오른쪽 사이드 메뉴를 확인한다.

■ 이번 과정에서는 직접 다운로드하여 로컬에 설치하여 실행한다.

- http://github.com/stepanowon/todosvc 에 방문하여 Readme.MD 의 내용을 참조하여 실행한다.
- 브라우저에서 http://localhost:8000 로 기능을 확인해본다.

2. 테스트용 API 소개(2)

- 브라우저에서 직접 조회한 결과
  - https://todosvc.herokuapp.com/todolist/gdhong
  - gdhong 사용자의 todolist 목록 조회



■ 그림처럼 나타내려면 크롬웹스토어에서 jsonview 확장 프로그램을 설치하면 된다.

2. 테스트용 API 소개(3)

todosvc API 목록

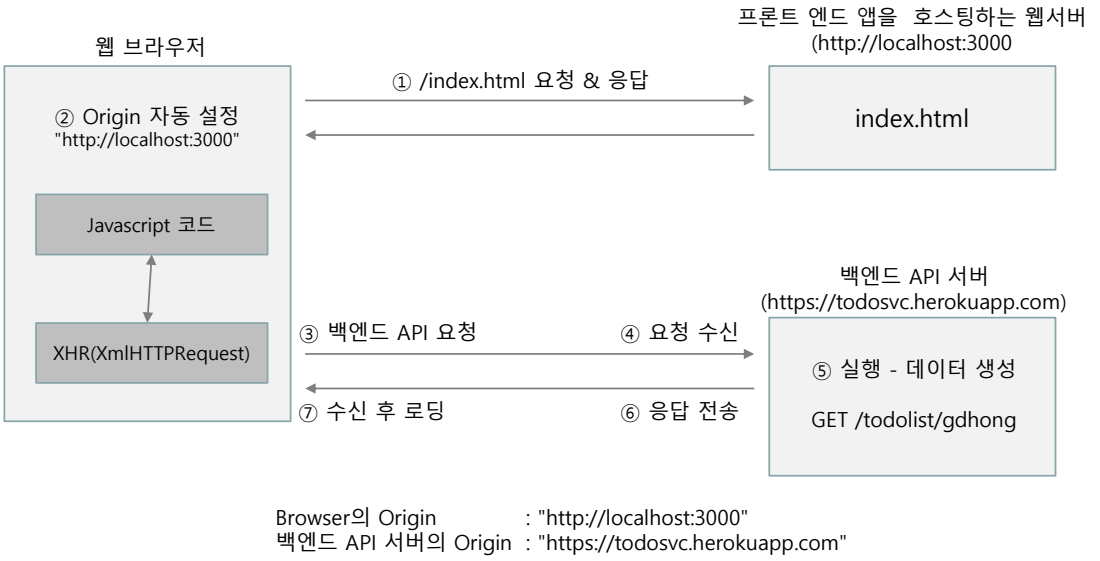
- GET /todolist/:owner
  - 특정 사용자(owner)의 todolist 조회
- GET /todolist/:owner/:id
  - 특정 사용자의 id에 해당하는 todo 한건 조회
- POST /todolist/:owner
  - 사용자의 todolist 새로운 todo 추가
- PUT /todolist/:owner/:id
  - 사용자의 id에 해당하는 todo 한건 수정
- PUT /todolist/:owner/:id/done
  - todo 한건의 완료 여부 toggle
- DELETE /todolist/:owner/:id
  - todo 한건 삭제
- GET /todolist/:owner/create
  - 새로운 사용자(owner)를 위한 샘플 todo 데이터 4건 생성

■ GET /todolist/:owner/create 를 제외한 나머지 엔드포인트들은 모두 /todolist 대신에 /todolist_long 경로로 요청이 가능함. 이 경로는 의도적인 1초의 지연시간을 발생시킴.

3. 크로스 도메인 문제 해결(1)

Cross Domain 문제란?

- SOP : Same Origin Policy, 브라우저의 기본 보안 정책



5

■ SOP는 브라우저의 기본 보안 정책으로 Consumer에서 로딩한 문서의 AJAX 기능은 동일 오리진이 아니라면 AJAX 요청을 하더라도 응답 데이터를 수신할 수 없도록 하는 보안 정책이다.

■ 오리진 정보 : "http://localhost:3000"

- 문자열이며, 이 문자열이 다르면 다른 오리진으로 간주한다.
- 따라서 동일 서버, 동일 주소라도 포트 번호가 다르면 다른 오리진이다.
- 동일 주소, 동일 포트라 할지라도 http, https 와 같이 리소스 종류가 다르다면 다른 오리진이다.

3. 크로스 도메인 문제 해결(2)

Cross Domain 문제 확인

- 의도적으로 Cross Domain 문제를 발생시켜 봄

프로젝트 생성

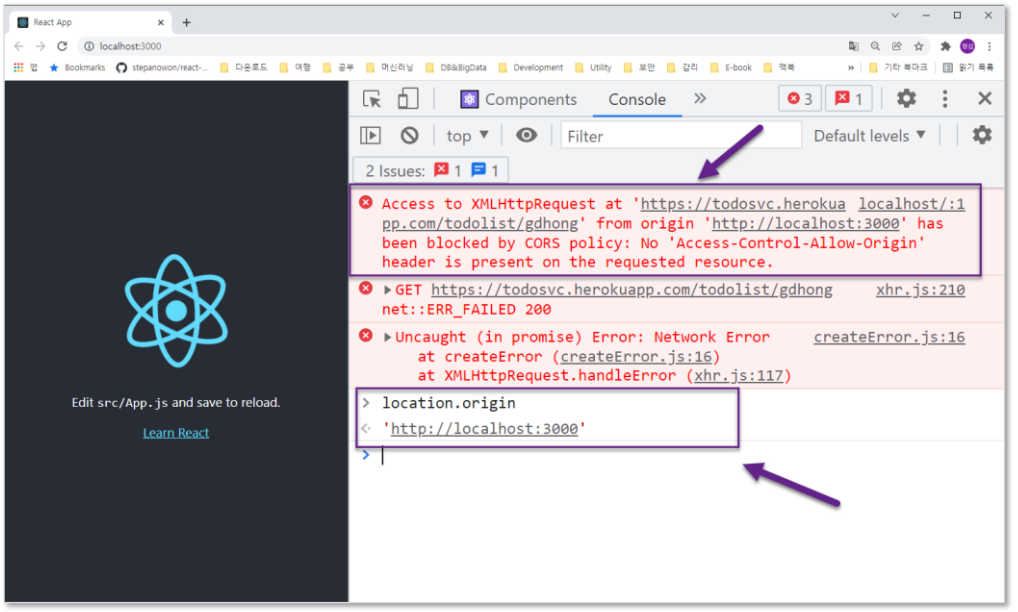
- npx create-react-app cross-domain-test
- cd cross-domain-test
- yarn add axios

src/App.js에 다음 코드 추가

```
……
import axios from 'axios';

const url = 'https://todosvc.herokuapp.com/todolist/gdhong';
axios.get(url)
.then((response)=>{
  console.log("## 응답 객체 : ", response)
})
……
```

3. 크로스 도메인 문제 해결(3)

Cross Domain 문제 발생 확인



■ 브라우저의 오리진 : http://localhost

- 브라우저 콘솔에서 location.origin 값으로 확인할 수 있음

■ 백엔드 API 서버의 오리진 : https://todosvc.herokuapp.com

3. 크로스 도메인 문제 해결(4)

크로스 도메인 문제 해결 방법

- 프론트엔드 앱을 호스팅하는 서버에 프록시 요소 생성
- 백엔드 API 서비스 제공자측에서 CORS기능을 제공

우리가 개발하는 것은 프론트엔드 애플리케이션

- CORS는 백엔드 API 서비스 제공자가 지원해야 함.
  - todosvc API는 필자가 CORS 지원을 하도록 설정하지 않았음
  - 백엔드 API 개발자들이 CORS를 반드시 지원한다는 보장이 없음
  - CORS 기능에 대한 자세한 내용은 다음을 참조한다.
    - https://developer.mozilla.org/ko/docs/Web/HTTP/Access_control_CORS
    - http://homoefficio.github.io/2015/07/21/Cross-Origin-Resource-Sharing/
- 프론트엔드 API 서비스에 프록시 요소를 설정할 수 있음
  - node.js 서버에 http-proxy-middleware 설정
  - 각 웹서버 관련 기술들이 대부분 proxy 기능 제공

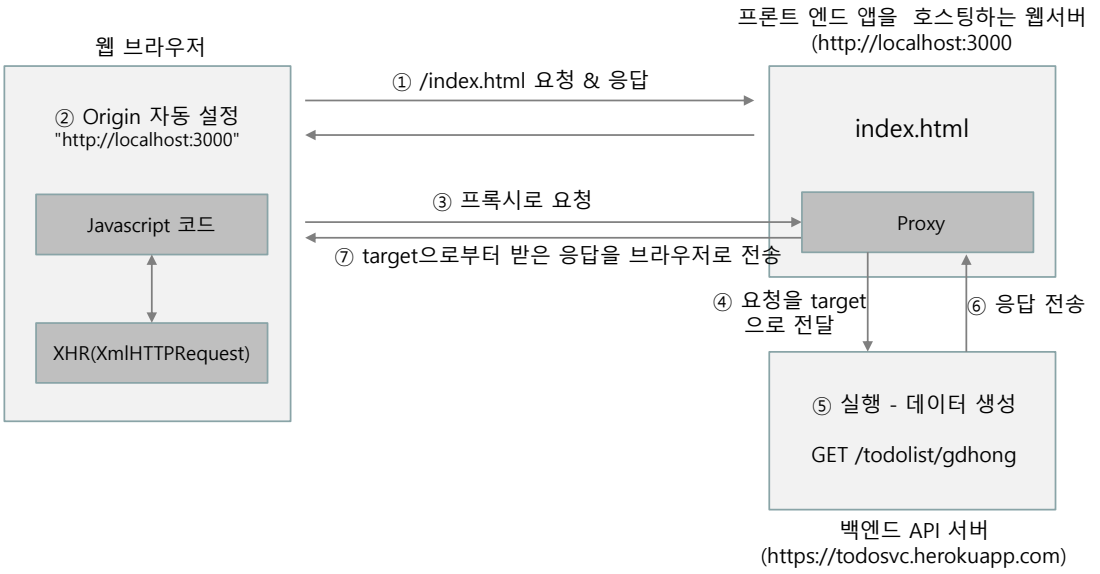
■ 프론트엔드 앱을 호스팅하는 웹서버와 백엔드 API 서버를 동일한 것으로 설정하면 Cross Domain 문제는 발생하지 않음. 하지만 리액트 앱을 개발하는 동안은 create-react-app이 제공하는 '개발용 서버'를 이용해야 하는데, 이 때는 크로스 도메인 상황임.

- 따라서 '개발용 서버'에서 프록시를 설정하는 방법을 반드시 알고 있어야 함.

3. 크로스 도메인 문제 해결(5)

create-react-app의 proxy 설정 기능

- setupProxy.js 작성 : 슬라이드 노트 참조



3. 크로스 도메인 문제 해결(6)

cross-domain-test 앱에 프록시 적용

- src/setupProxy.js 생성

```
const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {
    app.use(
        '/api',
        createProxyMiddleware({
            target: 'https://todosvc.herokuapp.com',
            changeOrigin: true,
            pathRewrite: {
                '^/api':''
            }
        })
    );
};
```

10

■ 프록시 설정에 대한 설명

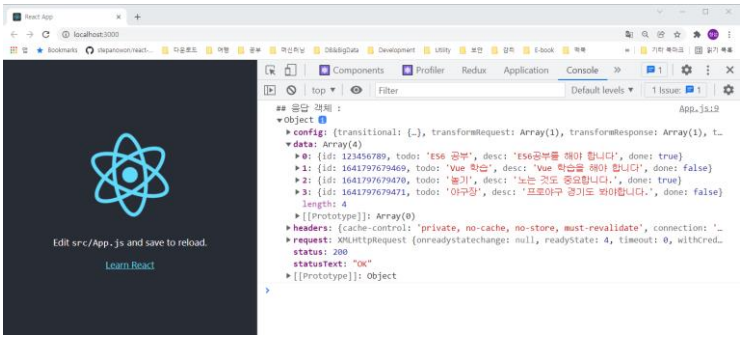
- /api 로 시작하는 경로로 요청하면 target으로 전달함
- pathRewrite 설정 : 경로롤 다시 쓴다는 의미
  - 정규표현식 사용
  - 요청 경로에서 '/api' 문자열은 '' 로 변경 : /api를 삭제함
  - 예) /api/todolist/gdhong 으로 요청하면 https://todosvc.herokuapp.com/todolist/gdhong 으로 전달함.

3. 크로스 도메인 문제 해결(7)

- src/App.js 변경

```
import axios from 'axios';

//const url = 'https://todosvc.herokuapp.com/todolist/gdhong';
const url = '/api/todolist/gdhong';
axios.get(url)
.then((response)=>{
  console.log("## 응답 객체 : ", response)
})
```

- 개발용 서버를 재시작하여 정상적인 응답을 확인



■ 다음 경로로 직접 요청하여 프록시의 기능을 확인해볼 수 있음

- http://localhost:3000/api/todolist/gdhong

4. axios 사용(1)

axios 사용법

- 라이브러리 참조
  - npm install --save axios (또는) yarn add axios
  - <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
- API 이용 방법

```
[저수준 API]
axios(config)
axios(url, config)

[각 메소드별 별칭]
axios.get(url[, config])
axios.delete(url[, config])
axios.post(url[, data[, config]])
axios.put(url[, data[, config]])
axios.head(url[, config])
axios.options(url[, config])
```


12

4. axios 사용(2)

- axios 저수준 메서드와 get 메서드

```
axios({
    method : 'GET',
    url : '/contacts',
    params : { pageno : 1, pagesize:5 }
})
.then((response) => {
    console.log(response);
})
.catch((error)=> {
    console.log("ERROR!!!! : ", error);
})
```

```
axios.get('/contacts', {
  params : { pageno:1, pagesize:5 }
})
.then(...)
.catch(...)
```

```
axios.get("https://todosvc.herokuapp.com/todolist/gdhong")
.then((response)=> {
    console.log(response);
})
.catch(……)
```

4. axios 사용(3)

- 응답 형식

```
▼Object {data: Object, status: 200, statusText: "OK", headers: Object, config: Object…}
  ▶config: Object
  ▶data: Object
  ▶headers: Object
  ▶request: XMLHttpRequest
   status: 200
   statusText: "OK"
  ▶__proto__: Object
```

- config : 요청 시에 사용된 config 옵션 정보입니다.
- headers : 응답 헤더 정보입니다.
- request : 서버와 통신 시에 사용된 XMLHttpRequest 객체 정보입니다.
- status : HTTP 상태 코드(HTTP Status Code)
- statusText : 서버 상태를 나타내는 문자열 정보입니다.

status code

- 2XX : 성공
- 3XX : 리다이렉션
- 4XX : 요청 오류(클라이언트측 오류)
- 5XX : 서버 오류

14

4. axios 사용(4)

- 기타 메서드

```
axios.post(BASEURL + '/todolist/user1',
    { todo:"독서하기", desc:"인문서적 1권 이번주까지" })
.then((response) => {
    if (response.data.status !== "success") {
        throw new Error("데이터 추가 실패!!");
    }
    console.log(response.data);
})
.catch(...)
```

```
axios.put(BASEURL + '/todolist/user1/1234',
    { todo:"독서하기", desc:"인문서적 2권 이번주까지", done:true })
.then((response) => {
    if (response.data.status !== "success") {
        throw new Error("데이터 변경 실패!!");
    }
    console.log(response.data);
})
.catch(...)
```

4. axios 사용(5)

- axios config 옵션
  - baseURL : 이 옵션을 이용해 공통적인 URL의 앞부분을 미리 등록해두면 요청 시 나머지 부분만을 요청 URL로 전달하면 됩니다. 가능하다면 axios.defaults.baseURL 값을 미리 바꾸는 편이 좋습니다.
  - transformRequest : 요청 데이터를 서버로 전송하기 전에 데이터를 변환하기 위한 함수를 등록합니다.
  - transformResponse : 응답 데이터를 수신한 직후에 데이터를 변환하기 위한 함수를 등록합니다.
  - headers : 요청시에 서버로 전달하고자 하는 HTTP 헤더 정보를 설정합니다.

```
// todolist_long 은 1초의 의도적 지연시간을 일으키는 엔드포인트임
axios.defaults.baseURL = '/api/todolist_long';
//인증 토큰은 백엔드 API 요청시 항상 전달하므로 기본값으로 설정할 수 있음
axios.defaults.headers.common['Authorization'] = JWT;
//timeout에 설정된 시간내에 응답이 오지 않으면 연결을 중단(abort)시킴
axios.defaults.timeout = 2000;
```


16

4. axios 사용(6)

axios default 설정

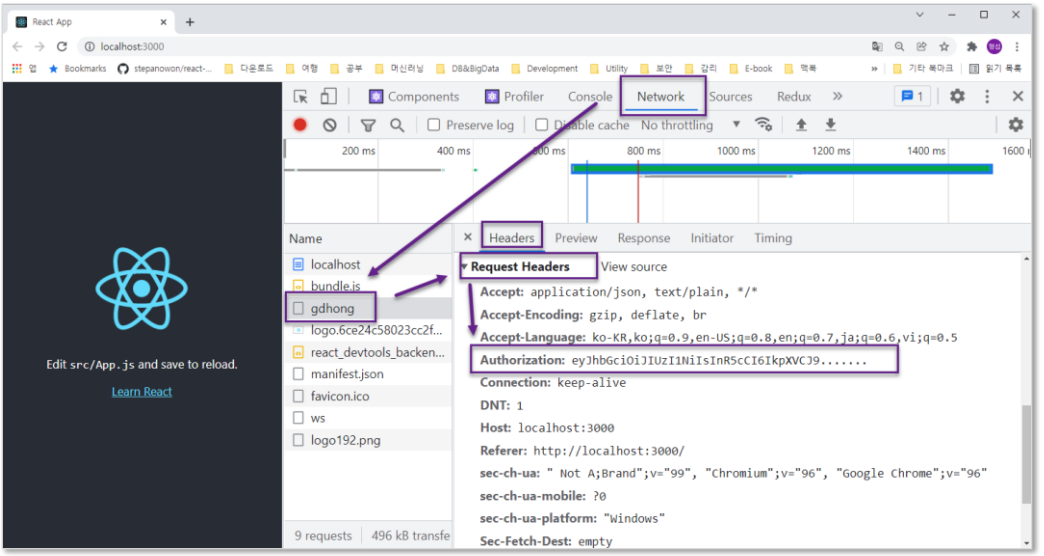
- src/App.js 변경

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

let TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.....";
axios.defaults.baseURL = '/api/todolist_long';
axios.defaults.headers.common['Authorization'] = TOKEN;
axios.defaults.timeout = 2000;

//const url = 'https://todosvc.herokuapp.com/todolist/gdhong';
//const url = '/api/todolist/gdhong';
const url = '/gdhong';    //baseURL 설정으로 인해 간단해짐
axios.get(url)
.then((response)=>{
  console.log("## 응답 객체 : ", response)
})
```


17

4. axios 사용(7)

실행 결과



18

4. axios 사용(8)

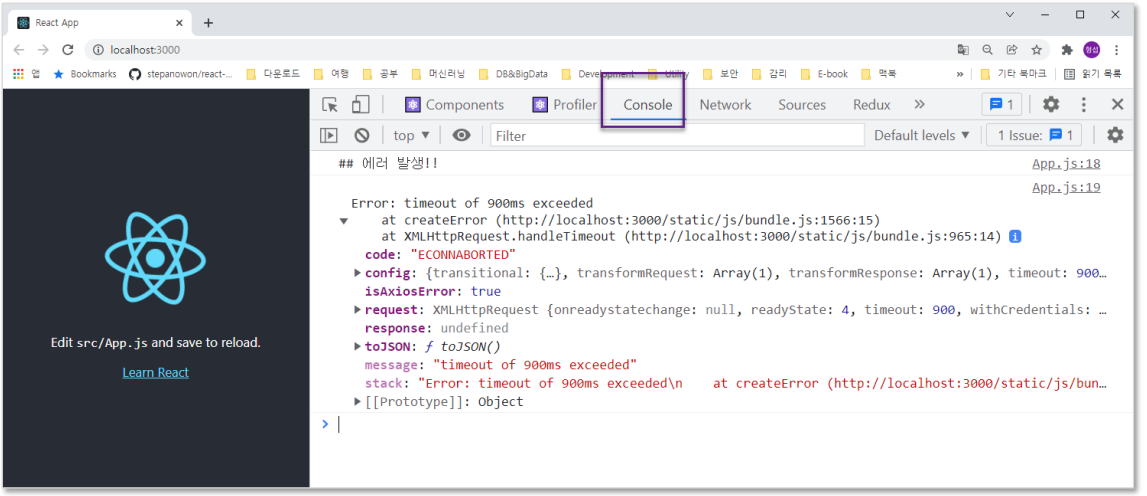
timeout을 1초의 지연시간보다 짧은 값으로 설정하면?

- timeout!! --> 오류 발생
- 이 오류는 Promise의 catch()로 처리함

```
.....
let TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.....";
axios.defaults.baseURL = '/api/todolist_long';
axios.defaults.headers.common['Authorization'] = TOKEN;
//1초(1000ms)보다 짧은 값으로 설정하여 오류 발생시킴
//오류는 catch()에 의해 처리됨
axios.defaults.timeout = 900;

const url = '/gdhong';    //baseURL 설정으로 인해 간단해짐
axios.get(url)
.then((response)=>{
  console.log("## 응답 객체 : ", response)
})
.catch((error)=>{
  console.log("##에러 발생!")
  console.dir(error)
})
```

4. axios 사용(9)

timeout 오류



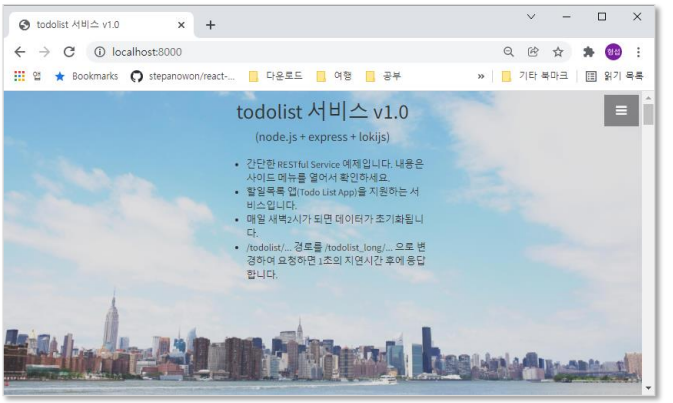
20

5. todolistapp에 axios 적용(1)

백엔드 API 서비스 구동

- 개개인마다의 독립성을 위해 자신의 컴퓨터에서 서버 구동
- https://github.com/stepanowon/todosvc 에서 코드 다운로드
- Readme.MD 파일의 내용을 참조하여 실행
  - npm run build
  - npm run start
  - http://localhost:8000

```
PS D:\workspace_temp\reactquick\todosvc> npm run start

> todosvc@1.0.0 start
> node build/index.js

할일 목록 서비스가 8000번 포트에서 시작되었습니다!
### GET /
### GET /todolist/:owner
### GET /todolist_long/:owner
```



21

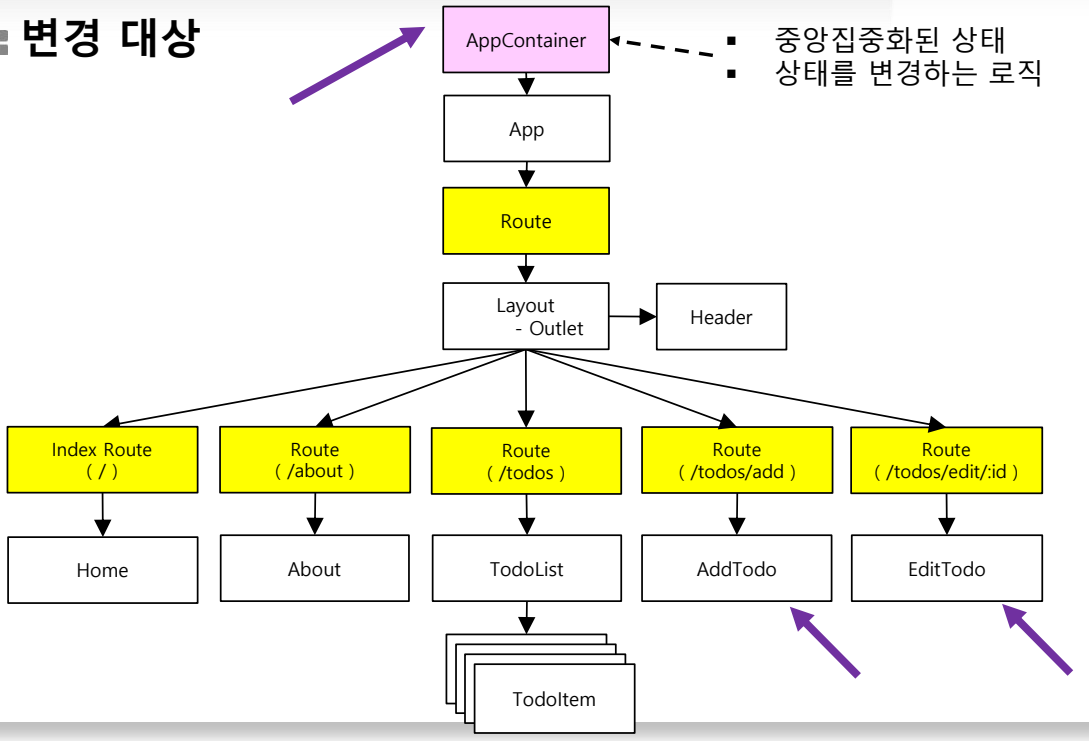
5. todolistapp에 axios 적용(2)

- 8장에서 작성한 todolistapp에 적용
- 환경 설정 추가
  - yarn add axios react-spinners
  - src/setupProxy.js 작성

```
const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {
    app.use(
        '/api',
        createProxyMiddleware({
            target: 'http://localhost:8000',
            changeOrigin: true,
            pathRewrite: {
                '^/api':''
            }
        })
    );
};
```

5. todolistapp에 axios 적용(3)

변경 대상



23

■ 변경사항

- AppContainer 컴포넌트
  - axios 를 이용해 todosvc 액세스하도록 코드 변경
- AddTodo, EditTodo
  - 할일의 추가, 수정이 확인된 후에 화면이 전환될 수 있도록 콜백 함수 적용

5. todolistapp에 axios 적용(4)

src/AppContainer.js 변경

```
import React, { useState, useEffect } from 'react';
import App from './App';
import produce from 'immer';
import axios from 'axios';

const USER = "gdhong";
const BASEURI = "/api/todolist/" + USER;

const AppContainer = () => {
    let [todolist, setTodoList] = useState([]);

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

    const fetchTodoList = () => {
        setTodoList([]);
        axios.get(BASEURI)
        .then((response)=> {
            setTodoList(response.data);
        })
    }
```

24

■ AppContainer 컴포넌트가 마운트될 때 초기 데이터를 미리 로드하도록 useEffect()를 사용할 수 있음.

- 또는 클래스 컴포넌트로 작성하고 componentDidMount() 생명주기 메서드를 이용해도 됨.

5. todolistapp에 axios 적용(5)

src/AppContainer.js 변경(이어서)

```
const addTodo = (todo, desc, callback) => {
    axios.post(BASEURI, { todo, desc })
    .then((response)=> {
        if (response.data.status === "success") {
            let newTodolist = produce(todolist, (draft)=> {
                draft.push(
                    { ...response.data.item, done:false })
            })
            setTodoList(newTodolist);
            callback();
        } else {
            alert("할일  추가 실패 : " +
                response.data.message);
        }
    })
    .catch((error)=> {
        alert("할일  추가 실패 : " + error);
    })
}
```

```
const deleteTodo = (id) => {
    axios.delete(`${BASEURI}/${id}`)
    .then((response)=>{
        if (response.data.status === "success") {
            let index = todolist.findIndex(
                (todo)=> todo.id === id);
            let newTodolist = produce(todolist, (draft)=> {
                draft.splice(index,1);
            })
            setTodoList(newTodolist);
        } else {
            alert("할일 삭제 실패 : " +
                response.data.message);
        }
    })
    .catch((error)=>{
        alert("할일  삭제 실패 : " + error);
    })
}
```

25

■ addTodo 메서드의 callback 인자는 todo 추가가 완료된 후 화면 전환이 이루어지도록 하기 위함.

- AddTodo, EditTodo 컴포넌트의 코드 변경 내용을 살펴보면 됨.

5. todolistapp에 axios 적용(6)

src/AppContainer.js 변경(이어서)

```
const toggleDone = (id) => {
    let todoitem = todolist.find((todo)=> todo.id === id);
    axios.put(`${BASEURI}/${id}`,
        { ...todoitem, done:!todoitem.done })
    .then((response)=>{
        if (response.data.status === "success") {
            let index = todolist.findIndex(
                (todo)=> todo.id === id);
            let newTodolist = produce(todolist, (draft)=> {
                draft[index].done = !draft[index].done;
            })
            setTodoList(newTodolist);
        } else {
            alert("할일  수정 실패 : " +
                response.data.message);
        }
    })
    .catch((error)=>{
        alert("할일  수정 실패 : " + error);
    })
}
```

```
const updateTodo = (id, todo, desc, done, callback) => {
    axios.put(`${BASEURI}/${id}`, { todo, desc, done })
    .then((response)=>{
        if (response.data.status === "success") {
            let index = todolist.findIndex(
                (todo)=> todo.id === id);
            let newTodolist = produce(todolist, (draft)=> {
                draft[index] =
                    { ...draft[index], todo, desc, done }
            })
            setTodoList(newTodolist);
            callback();
        } else {
            alert("할일  수정 실패 : " +
                response.data.message);
        }
    })
    .catch((error)=>{
        alert("할일  수정 실패 : " + error);
    })
}
```

5. todolistapp에 axios 적용(7)

src/AppContainer.js 변경(이어서)

```
    const getTodoOne = (id) => todolist.find((todo)=>todo.id === parseInt(id,10));

    const callbacks = { addTodo, deleteTodo, updateTodo, toggleDone, getTodoOne, fetchTodoList };
    const states = { todolist };

    return (
        <App callbacks={callbacks} states={states} />
    );
};

export default AppContainer;
```

- 할일 추가후 화면의 전환에 대해서
  - 할일 추가가 성공했음을 백엔드 서버로부터 응답받은 후 화면이 전환되지 않고 그전에 전환되도록 작성되어 있었음
  - 이를 위해 콜백 함수를 전달하는 방법 사용
    - EditTodo, AddTodo에 적용

5. todolistapp에 axios 적용(8)

src/pages/AddTodo.js, EditTodo.js 변경

- 추가, 변경이 완료된 후 다른 경로로 이동될 수 있도록 callback 메서드를 전달함.

```
.....
const AddTodo = ({callbacks}) => {
    .....
    const addContactHandler = ()=> {
        if (todo.trim() === "" || desc.trim()==="") {
            alert('반드시 할일, 설명을 입력해야 합니다.');
            return;
        }
        callbacks.addTodo(todo, desc, ()=> {
            navigate('/todos');
        });
    }
    .....
};
```

```
.....
const EditTodo = ({ callbacks }) => {
    .....
    const updateContactHandler = ()=> {
        if (todoOne.todo.trim() === "" ||
            todoOne.desc.trim()==="") {
            alert('반드시 할일, 설명을 입력해야 합니다.');
            return;
        }
        let { id, todo, desc, done } = todoOne;
        callbacks.updateTodo(id, todo, desc, done, ()=> {
            navigate('/todos');
        });
    }
    .....
};
```

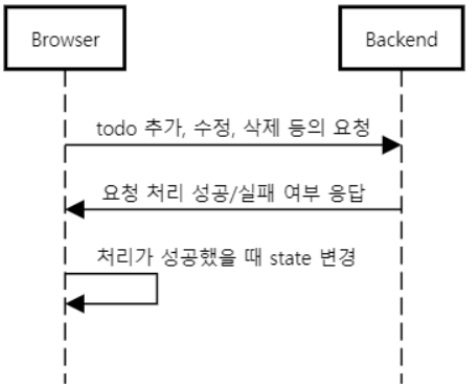
28

■ 일단 여기까지 작성한 결과를 실행해서 확인해볼 수 있음.

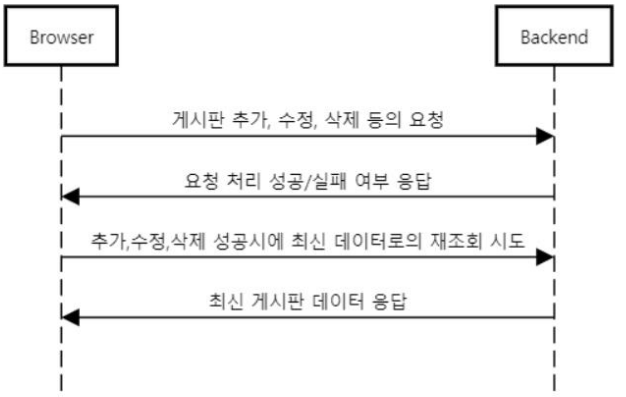
5. todolistapp에 axios 적용(9)

2가지 처리방식

ex)개인화된 데이터



ex)공용 데이터



5. todolistapp에 axios 적용(10)

지연 시간에 대한 처리

- 지연 시간 발생시 처리중을 의미하는 Spinner UI를 적용해야 함.
  - 8장의 Lazy Loading에서 다룬 Suspense를 생각할 수 있지만 axios로 요청하고 응답받는 상황은 라우팅하는 것이 아님.
  - 따라서 별도의 상태(State)와 UI를 구성해야 함.

src/components/Loading.js 컴포넌트 작성

```
import React from 'react';
import { ScaleLoader } from 'react-spinners';

const Loading = () => {
    return (
        <div className="bg-white w-100 h-100 position-fixed" style={{ top:0, left:0, opacity:0.8 }}>
            <div className="row w-100 h-100 justify-content-center align-items-center">
                <div className="col-6 text-center">
                    <h3>처리중</h3>
                    <ScaleLoader height="40px" width="6px" radius="2px" margin="2px" />
                </div>
            </div>
        </div>
    );
};
export default Loading;
```

30

5. todolistapp에 axios 적용(11)

src/AppContainer.js 변경

- BASEURI를 의도적 지연시간을 발생시키는 경로로 변경함.
- addTodo와 동일한 방법으로 deleteTodo, toggleDone, updateTodo를 변경함. (bold 부분을 확인)

```
……
const BASEURI = "/api/todolist_long/" + USER;

const AppContainer = () => {
  let [todolist, setTodoList] = useState([]);
  let [isLoading, setIsLoading] = useState(false);
  ……
  const fetchTodoList = () => {
    setTodoList([]);
    setIsLoading(true);
    axios.get(BASEURI)
    .then((response)=> {
      setTodoList(response.data);
      setIsLoading(false);
    })
  }
```

```
const addTodo = (todo, desc, callback) => {
  setIsLoading(true);
  axios.post(BASEURI, { todo, desc })
  .then((response)=> {
    if (response.data.status === "success") {
      let newTodolist = produce(todolist, (draft)=> {
        draft.push({ ...response.data.item, done:false })
      })
      setTodoList(newTodolist);
      callback();
    } else {
      alert("할일  추가 실패 : " + response.data.message);
    }
    setIsLoading(false);
  })
  .catch((error)=> {
    setIsLoading(false);
    alert("할일  추가 실패 : " + error);
  })
}
```

■ AppContainer의 마지막 부분에서 states에 isLoading을 추가하고, callbacks에 fetchTodoList를 추가한다.

```
……
const callbacks = { addTodo, deleteTodo, updateTodo, toggleDone, getTodoOne,
fetchTodoList };
const states = { todolist, isLoading };

return (
    <App callbacks={callbacks} states={states} />
);
};

export default AppContainer;
```

5. todolistapp에 axios 적용(12)

src/App.js 변경

- props로 전달받은 isLoading 상태값에 따라 <Loading /> 컴포넌트의 렌더링 여부 결정함.

```
……
import Loading from './components/Loading';

const App = ({ states,callbacks }) => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Layout />}>
            ……
        </Route>
      </Routes>
      { states.isLoading ? <Loading /> : "" }
    </Router>
  );
};
……
```


32

5. todolistapp에 axios 적용(13)

src/pages/TodoList.js 변경

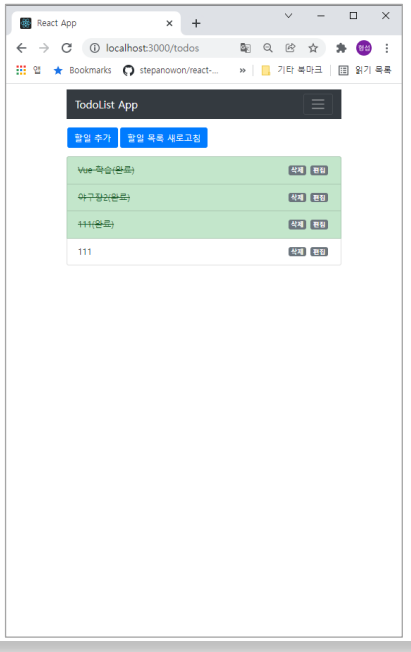
- 할일 새로 고침 버튼 추가

```
……
const TodoList = (props) => {
    ……
    return (
        <>
        <div className="row">
            <div className="col p-3">
                <Link className="btn btn-primary mr-1" to="/todos/add">할일 추가</Link>
                <button className="btn btn-primary mr-1"
                    onClick={ ()=>props.callbacks.fetchTodoList() }>할일 목록 새로고침</button>
            </div>
        </div>
        <div className="row">
            ……
        </div>
        </>
    );
};
……
```


33

5. todolistapp에 axios 적용(14)

실행 결과 확인



34