ToDo

개요

- ToDo Application

개요

- ToDo Application
  - ✓ 구현 기능
    - ToDo 생성: + 버튼을 클릭해 ToDo 아이템을 생성
    - ToDo 리스트: 생성된 아이 템 목록을 화면에서 확인
    - ToDo 수정: ToDo 아이템을 체크하거나 내용을 수정
    - ToDo 삭제: ToDo 아이템을 삭제

개요

- ToDo Application
  - ✓ 기술
    - React.js
      - FrontEnd 애플리케이션 개발에 사용
    - MySQL
      - 반 영구적으로 데이터를 저장하기 위한 관계형 데이터베이스
    - Python Django
      - BackEnd 애플리케이션 개발에 사용
      - Spring Boot로 REST API를 구현
      - API는 FrontEnd 애플리케이션이 사용
      - REST API를 구현하고 FrontEnd를 분리하는 것은 마이크로 서비스 아키텍처로 서비스를 확장하는 데 용이

개요

- ❖ 개발 환경
  - ✓ Back End
    - Python3
    - MySQL 5.0 이상 버전
  - ✓ FrontEnd
    - Visual Studio Code: IDE
    - Node

Back End

- MySQL
  - ✓ 데이터베이스에 접속해서 데이터베이스 생성

    create database adam;

Back End

- Back End Project 생성 및 설정
  - ✓ 프로젝트 디렉토리 생성
  - ✓ 가상환경 생성
    - python -m venv myvenv
    - myvenv₩Scripts₩activate(source myvenv/bin/activate)
  - ✓ 패키지 설치
    - pip install django mysqlclient djangorestframework
  - ✓ 프로젝트 생성
    - django-admin startproject todoproject .
  - ✓ 애플리케이션 생성
    - python manage.py startapp todoapplication

Back End

- Python Project 생성 및 설정
  - ✓ 프로젝트 내의 settings.py 수정

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "rest_framework",
    "todoapplication",
]
```

Back End

- Python Project 생성 및 설정
  - ✓ 프로젝트 내의 settings.py 수정

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "adam",
        "USER": "root",
        "PASSWORD": "wnddkd",
        "HOST": "127.0.0.1",
        "PORT": "3306"
    }
}
```

Back End

- Python Project 생성 및 설정
  - ✓ 프로젝트 내의 settings.py 수정

    TIME_ZONE = "Asia/Seoul"

Back End

- Model 생성
  - ✓ 애플리케이션의 models.py 파일 작성

```
from django.db import models

class Todo(models.Model):
    id = models.AutoField(primary_key=True)
    userid=models.CharField(max_length=100)
    title=models.CharField(max_length=100)
    done= models.BooleanField(default=False)
    regdate = models.DateTimeField(auto_now_add=True)
    moddate = models.DateTimeField(null=True)
```

Back End

- ❖ Model 생성
  - ✓ migration
    - python manage.py makemigrations
    - python manage.py migrate

Back End

- Model 생성
  - ✓ 확인
    - show tables
    - desc todoapplication_todo

auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
django_admin_log
django_content_type
django_migrations
django_session
example_book
myweb_item
todo_todo
todoapplication_todo

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	[NULL]	auto_increment
userid	varchar(100)	NO		[NULL]	
title	varchar(100)	NO		[NULL]	
done	tinyint(1)	NO		[NULL]	
regdate	datetime(6)	NO		[NULL]	
moddate	datetime(6)	NO		[NULL]	

Back End

- CRUD 작업
  - ✓ 프로젝트의 urls.py 파일 수정

```
from django.contrib import admin
from django.urls import path
from todoapplication import views

urlpatterns = [
    path("admin/", admin.site.urls),
    path("todo", views.TodoView.as_view()),
]
```

Back End

- CRUD 작업
  - ✓ 애플리케이션의 views.py 파일 수정

```
from django.views import View
from django.http import JsonResponse

from rest_framework import status

from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator

from .models import Todo

from datetime import datetime
import json

#Todo 인스턴스를 dict로 변환해주는 함수
def todoToDictionary(todo:Todo) -> dict:
    result = {"id":todo.id, "userid":todo.userid, "title":todo.title,
              "done":todo.done, "regdate":todo.regdate, "moddate":todo.moddate}
    return result
```

Back End

- CRUD 작업
  - ✓ 애플리케이션의 views.py 파일 수정

```
#csrf 설정
@method_decorator(csrf_exempt, name='dispatch')
class TodoView(View):
    def post(self, request):
        request = json.loads(request.body)
        userid = request["userid"]
        title = request["title"]

        todo = Todo()
        todo.userid = userid
        todo.title = title
        todo.done = False

        todo.save()

        if userid != None:
            todos = Todo.objects.filter(userid=userid)
        else:
            todos = Todo.objects.all()
        return JsonResponse({'list': list(todos.values())}, status=status.HTTP_200_OK)
```

Back End

- CRUD 작업
  - ✓ 애플리케이션의 views.py 파일 수정

```
def get(self, request):
    userid = request.GET.get("userid", None)
    if userid != None:
        todos = Todo.objects.filter(userid=userid)
    else:
        todos = Todo.objects.all()
    return JsonResponse({'list':list(todos.values())}, status=status.HTTP_200_OK)
```

Back End

- CRUD 작업
  - ✓ 애플리케이션의 views.py 파일 수정

```
def put(self, request):
    request = json.loads(request.body)

    userid = request['userid']
    id = request["id"]
    done = request["done"]

    todo = Todo.objects.get(id=id)
    if todo.userid == userid:
        todo.done = done
        todo.moddate = datetime.today()

        todo.save()
        return JsonResponse({'result':True, 'data': todoToDictionary(todo)},
            status=status.HTTP_200_OK)
    else:
        return JsonResponse({'result':False, 'data': '수정 권한이 없습니다.'},
            status=status.HTTP_200_OK)
```

Back End

- CRUD 작업
  - ✓ 애플리케이션의 views.py 파일 수정

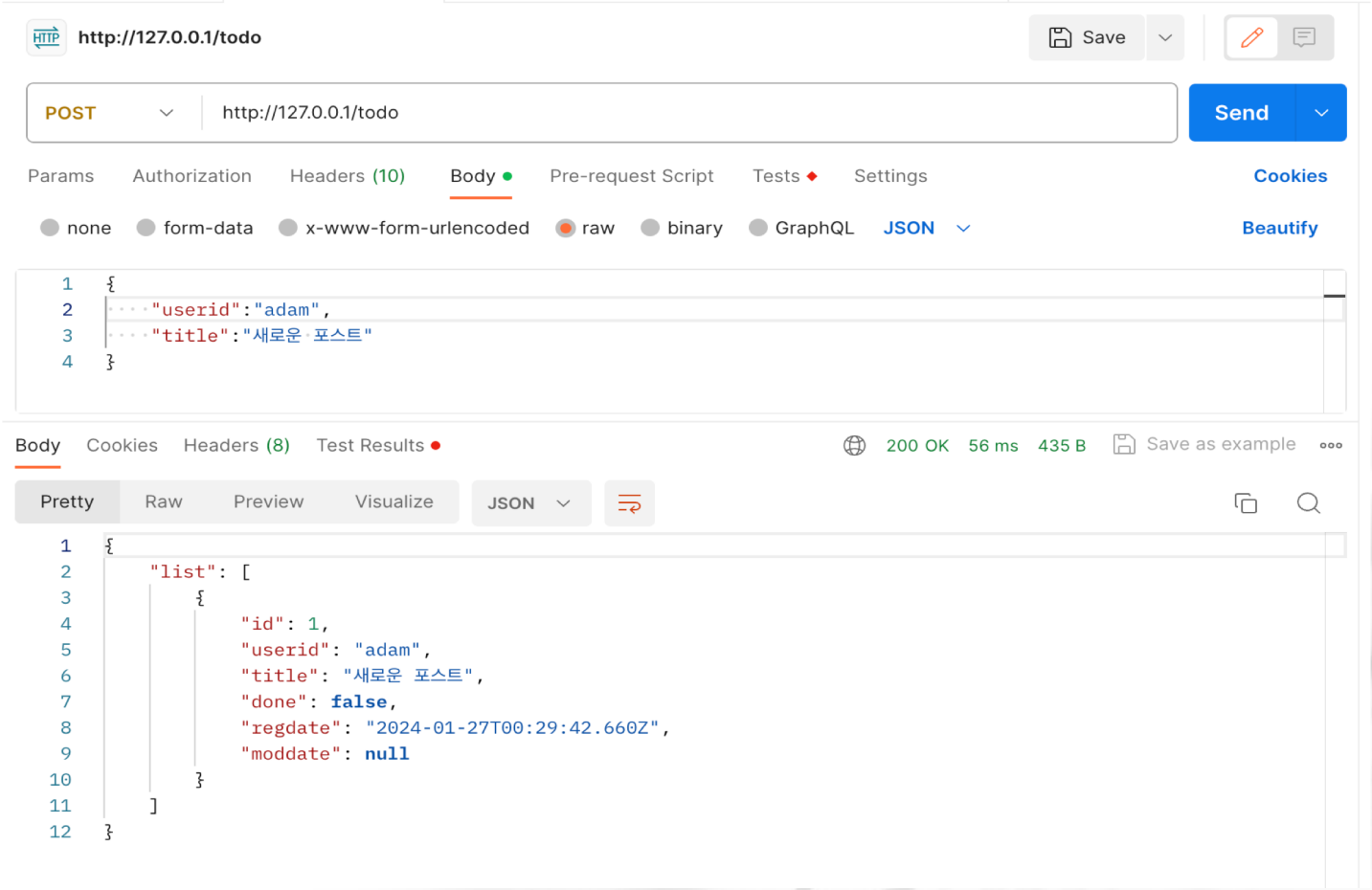
```
def delete(self, request):
    request = json.loads(request.body)

    userid = request['userid']
    id = request["id"]

    todo = Todo.objects.get(id=id)
    if todo.userid == userid:
        todo.delete()
        return JsonResponse({'result':True},
            status=status.HTTP_200_OK)
    else:
        return JsonResponse({'result':False},
            status=status.HTTP_200_OK)
```

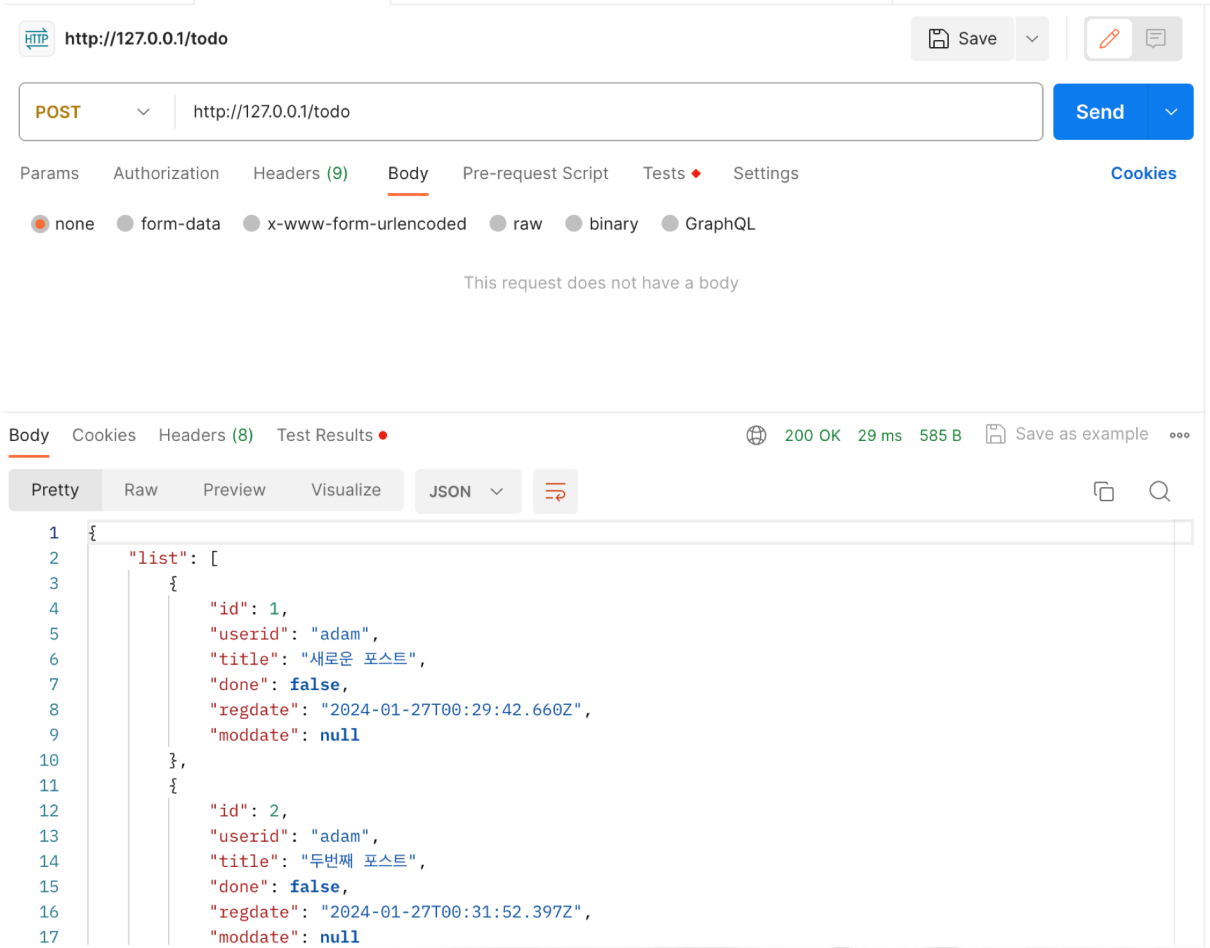
Back End

- 테스트 – PostMan
  - ✓ 데이터 삽입



Back End

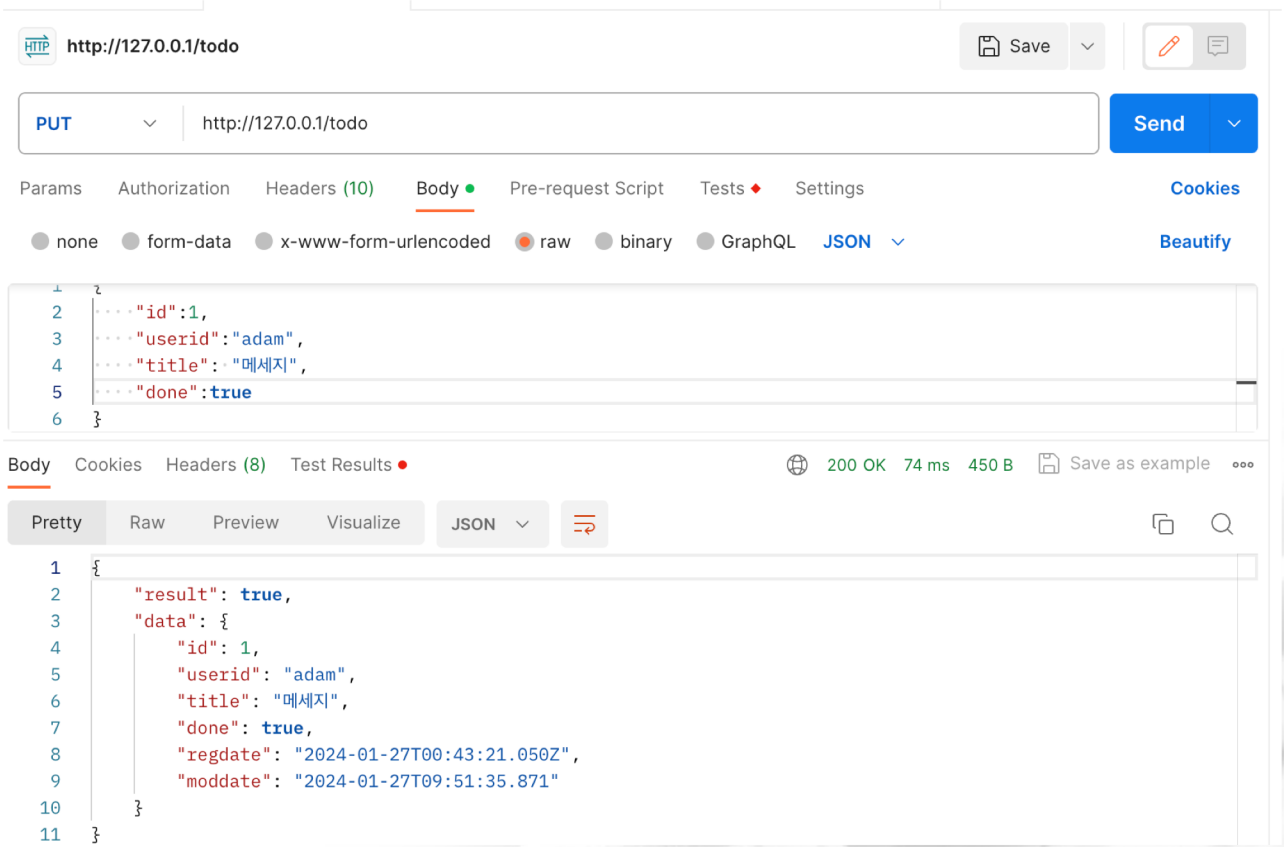
- ❖ 테스트 – PostMan
  - ✓ 데이터 조회



Back End

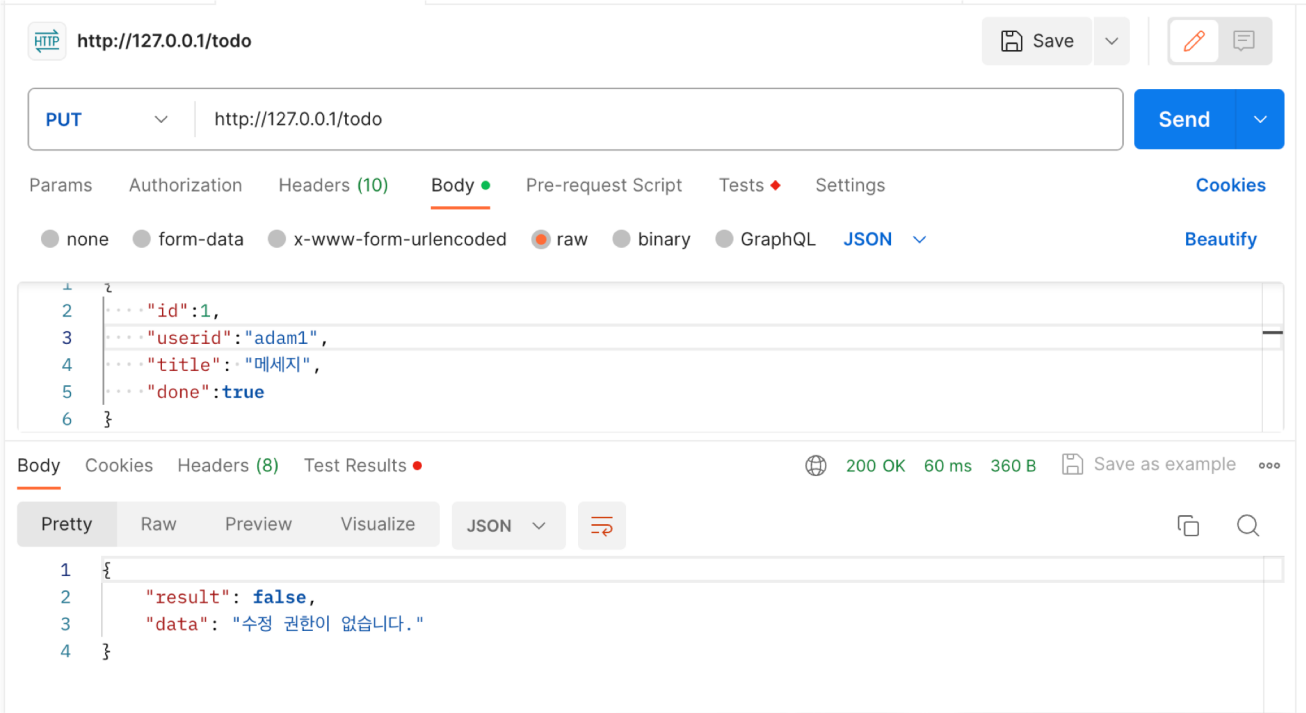
❖ 테스트 – PostMan

✓ 데이터 수정



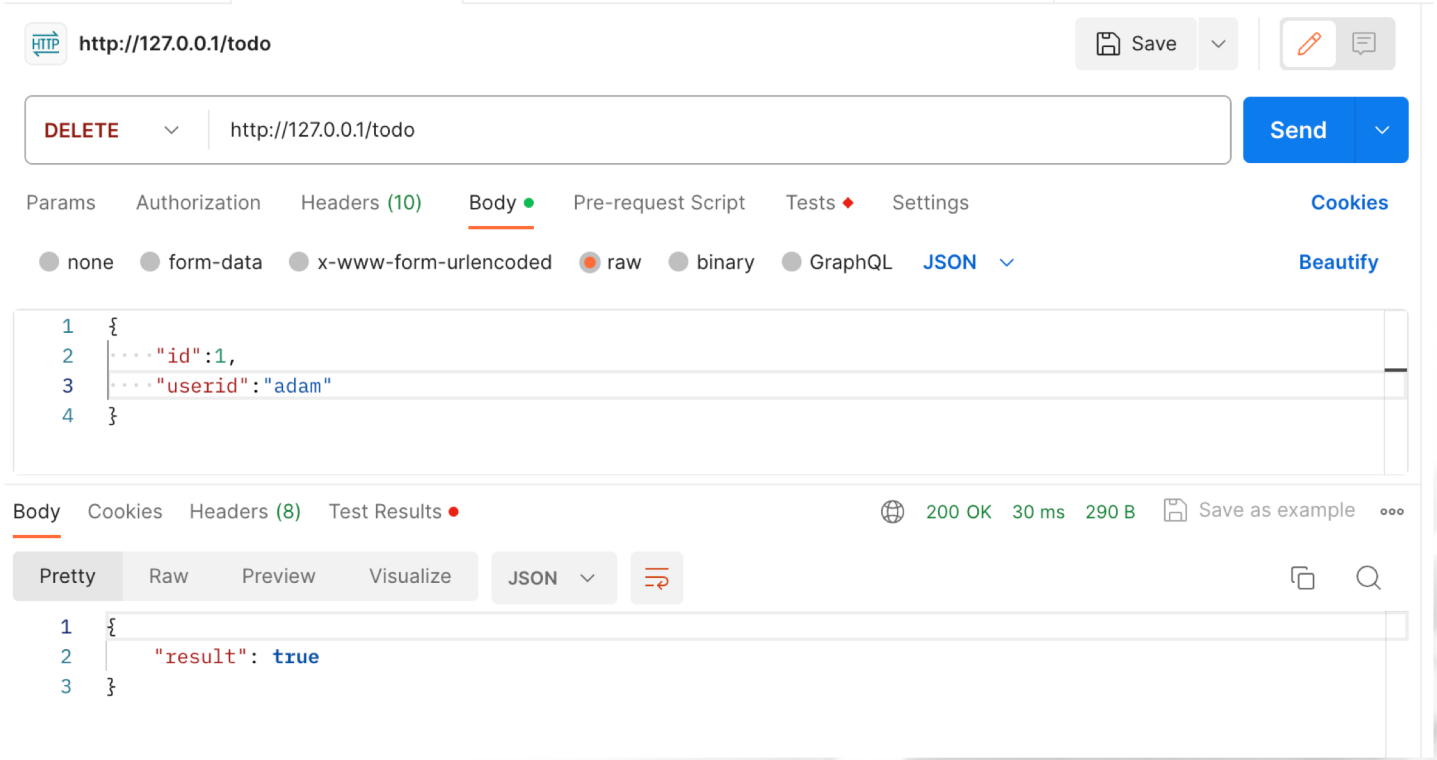
Back End

- ❖ 테스트 – PostMan
  - ✓ 데이터 수정



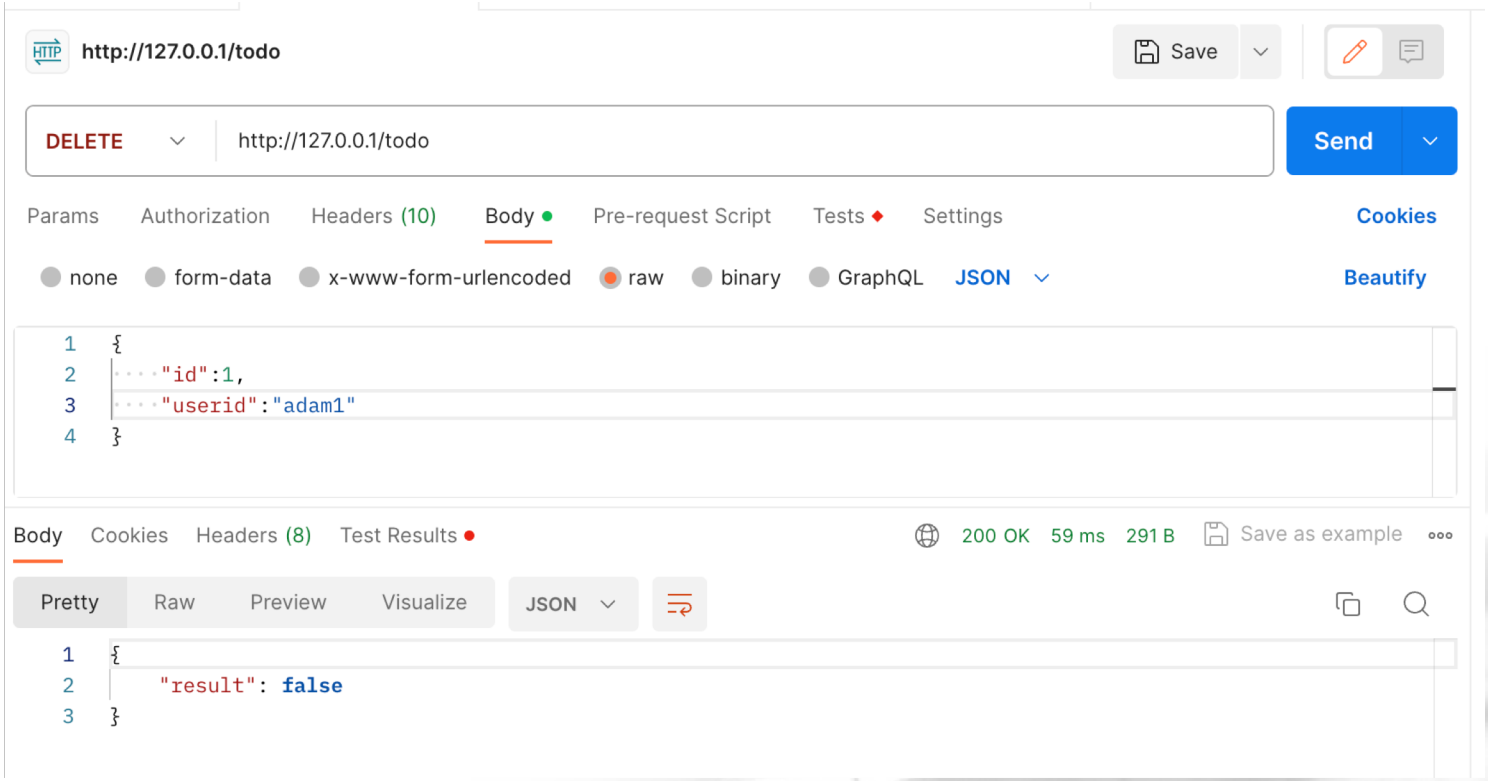
Back End

- 테스트 – PostMan
  - ✓ 데이터 삭제



Back End

- 테스트 – PostMan
  - ✓ 데이터 삭제



Front End

- SPA
  - ✓ 웹 서버는 웹 브라우저가 요청하는 다양한 유형의 자원을 제공하는 역할을 수행
  - ✓ 웹 브라우저에서 웹 서버가 제공하는 자원을 얻으려면 주소 창에 http://웹_서버/자원1 과 같은 URL 문자열을 HTTP 프로토콜을 사용하여 웹 서버에 요청해야 하는데 이를 HTTP 요청이라고 하고 HTTP 요청을 받은 웹 서버는 해당 자원을 웹 브라우저에 보내 주는데 이를 HTTP 응답이라고 하고 웹 브라우저는 웹 서버에서 보내온 HTTP 응답 데이터를 사용자가 볼 수 있도록 웹 페이지 화면에 보여 주는데 이를 렌더링(rendering)이라고 함
  - ✓ 어떤 웹 페이지가 처음 응답받은 자원을 렌더링한 뒤 다시 다른 자원을 요청하면 과거에 렌더링 한 내용을 모두 지우고 새로 수신한 자원을 렌더링하는데 이 과정에서 웹 페이지는 약간의 깜박임(새로 고침)이 생김
  - ✓ 웹 브라우저에서 주소 창으로 다양한 자원을 요청하는 방식으로 동작하는 웹 서버는 이런 깜박임 현상을 피할 수 없는데 주소 창으로 요청하는 자원이 하나뿐이라면 이런 깜박임은 일어나지 않음
  - ✓ 리액트 프레임워크로 만드는 웹 애플리케이션은 index.html 파일 1개로 동작하기 때문에 웹 서버에 자원을 한 번만 요청하므로 화면 깜박임 현상이 발생하지 않음
  - ✓ 리액트 프레임워크는 백엔드에서 받은 JSON 데이터를 해석하여 현재 화면에서 사용자가 새로 요청한 부분만 동적으로 화면을 생성하는데 이런 방식의 웹 애플리케이션을 싱글 페이지 애플리케이션을 SPA하고 하고 사용자 요청이 있을 때마다 완전히 새로운 HTML을 전달받는 기존 방식을 멀티 페이지 애플리케이션(Multip Page application – MPA)이라고 함
  - ✓ React 와 Vue는 싱글 페이지 애플리케이션을 만드는 프론트엔드 자바스크립트 프레임워크

Front End

- 템플릿 엔진
  - ✓ 웹 서버는 대부분 HTML 템플릿 엔진을 제공
  - ✓ 템플릿 엔진은 HTML 문서를 서버에서 처리한 결과 데이터와 결합하여 쉽게 만들어 주는데 프론트엔드에도 템플릿 엔진이 필요
  - ✓ 프론트엔드는 백엔드에서 제공하는 JSON 데이터를 해석하여 자바스크립트 객체들의 조합을 얻은 다음 이 객체들을 다시 웹 브라우저가 이해할 수 있는 D0M 객체로 변환해 주어야 함
  - ✓ 서버 쪽 템플릿 엔진의 출력물은 HTML이지만 프론트엔드 쪽 템플릿 엔진의 출력물은 DOM 객체들의 조합이라는 차이만 있을 뿐 데이터와 템플릿을 조합하여 출력물을 만들어 낸다는 동작 원리는 같음
  - ✓ 리액트, 앵귤러, 뷰 등 프론트엔드 프레임워크는 자바스크립트 객체를 DOM 객체로 전환해 주는 역할을 수행한다는 공통점이 있음
  - ✓ 프론트엔드 프레임워크는 클라이언트(웹 브라우저)에서 동작하는 템플릿 엔진

Front End

- React
  - ✓ React는 2013년에 페이스북에서 발표한 오픈소스 자바스크립트 프레임워크
  - ✓ 유저 인터페이스를 만드는 데 사용할 수 있는 자바스크립트 라이브러리
  - ✓ 리액트 프레임워크는 가상 DOM(Document Object Model) 과 JSX(JavaScript XML)라는 새로운 방식으로 동작하는 프레임워크
  - ✓ 자바스크립트의 특정 값이 바뀌면 특정 DOM의 속성이 바뀌도록 연결을 해주어서 업데이트하는 작업을 간소화해주는 방식으로 웹 개발의 어려움을 해결한 라이브러리
  - ✓ react 프로젝트에서 특정 부분이 어떻게 생길지 정하는 선언체가 있는데 이것을 컴포넌트(component) 라고 부름
  - ✓ 컴포넌트는 템플릿과는 다른 개념으로 템플릿은 보통 데이터 셋이 주어지면 HTML 태그 형식을 문자열로 반환하는데 이와 달리 컴포넌트는 조금 더 복합적인 개념인데 재사용이 가능한 API로 수 많은 기능들을 내장
  - ✓ 게임 엔진에 사용하는 방식인 다음에 나타날 화면의 일부(노드)를 미리 그려 놓고 변경된 화면의 일부(노드)만 수정하는 가상 화면(Virtual DOM) 기술을 이용
  - ✓ Virtual DOM 은 가상의 DOM으로 브라우저에 실제로 보여지는 DOM 이 아니라 메모리에 가상으로 존재하는 DOM 으로 JavaScript 객체이기 때문에 작동 성능이 브라우저에서 DOM을 보여주는 것 보다 속도가 빠름

Front End

- ❖ react 작업 환경
  - ✓ node.js
    - 노드 설치: https://nodejs.org/ko/download/
    - 설치 확인: node --version
    - Webpack 과 Babel 과 같은 도구들이 자바스크립트 런타임인 Node.js 기반으로 만들어져 있음
    - npm
      - ◆ https://www.npmjs.com 에서 필요한 라이브러리를 내려받아 설치하고 삭제하는 등의 관리를 해주는 프로그램으로 node를 설치하면 자동으로 설치 됨
      - ◆ node_modules 디렉토리에 라이브러리를 내려받아 저장하고 package.json 이라는 파일에 설치된 라이브러리의 정보를 기재해서 관리
      - ◆ 실제 라이브러리와 라이브러리 명세 파일을 따로 관리하는 이유는 node_modules에 저장되는 라이브러리의 용량이 크기 때문
    - yarn
      - ◆ npm을 개선한 프로그램
      - ◆ 더 나은 속도 와 더 나은 캐싱 시스템을 사용하기 위해서 사용
      - ◆ npm install --location=global yarn
      - ◆ 설치 확인: yarn --version

Front End

- ❖ react 작업 환경
  - ✓ node.js
    - webpack
      - ◆ 프로젝트에 사용된 파일을 분석하여 웹 문서 파일로 변환하는 Module Bundler
      - ◆ Webpack이 필요한 이유는 프레임워크가 .js, .css, .jpg 와 같은 기존 웹 문서 파일을 사용하지 않기 때문
      - ◆ 트위터 부트스트랩 템플릿은 웹 문서 스타일을 .css가 아닌 .sass 파일로 작성하는데 웹 브라우저는 .sass 파일을 해석하지 못하므로 중간에 누군가 이 파일을 해석해 주어야 하는데 이 역할을 하는 도구가 webpack
    - Babel
      - ◆ Babel은 ECMAScript 2015+ 코드를 이전 JavaScript 엔진에서 실행할 수 있는 이전 버전과 호환되는 JavaScript 버전으로 변환하는 데 사용되는 무료 오픈 소스 JavaScript 트랜스 컴파일러

Front End

- react 작업 환경
  - ✓ IDE – Visual Studio Code
    - https://code.visualstudio.com/
    - 플러그 인
      - Prettier: 자동으로 코드의 스타일을 관리해주는 도구
      - ESLint: 자바스크립트 문법을 확인해주는 도구
      - Relative Path(상대 경로에 있는 파일 경로를 편리하게 작성하는 도구)
      - Guides(들여쓰기 가이드 라인을 그려주는 도구)
      - reacts code snippets: 자주 사용되는 코드에 대해서 단축어를 만들어서 코드를 빠르게 생성해내는 것
      - Tailwind CSS
      - HeadWind: 테일윈드CSS 클래스 분류기
      - Post CSS: CSS 구문 강조 표시
      - 터미널에서 typescript 컴파일 설치
        - 설치: npm i -g typescript ts-node
        - 확인: tsc –v
        - 확인: ts-node -v

Front End

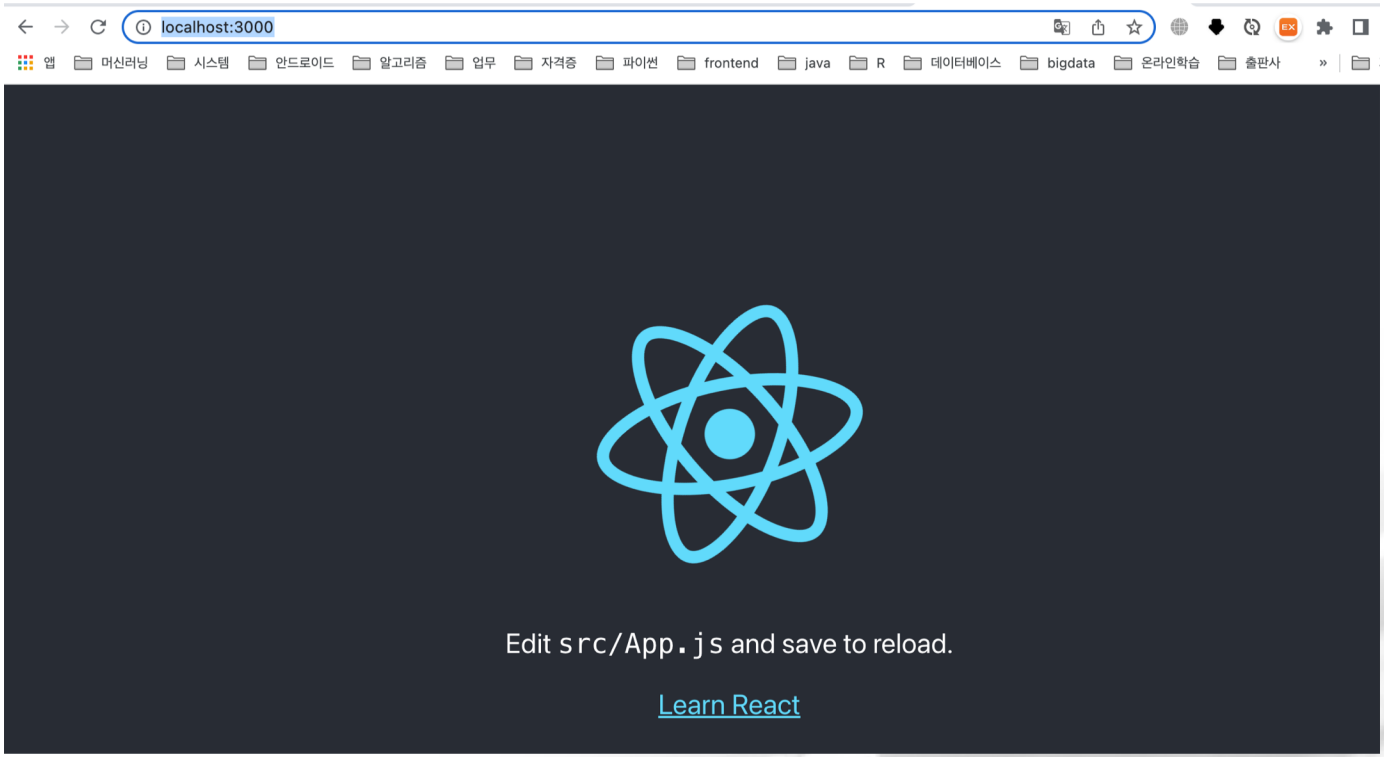
- ❖ 프로젝트 생성 및 실행
  - ✓ react 프로젝트 생성: todo-react-app 이 프로젝트 이름

    yarn create react-app todo-react-app
  - ✓ 애플리케이션 실행

    npm start(yarn start 도 가능)

Front End

- ❖ 프로젝트 생성 및 실행
  - ✓ 브라우저 확인



Front End

- UI 개발을 빠르게 하기 위한 material-ui 패키지
  - ✓ https://mui.com/
  - ✓ 설치

```
npm install --save --legacy-peer-deps @material-ui/core
npm install --save --legacy-peer-deps @material-ui/icons
```

Front End

- ToDo 목록 보기
  - ✓ src 디렉토리에 ToDo.jsx 파일을 추가하고 작성

```
import React from "react";

class ToDo extends React.Component {

  render() {
    return (
        <div className="ToDo">
            <input type="checkbox" id="todo0" name="todo0" value="todo0" />
            <lable for="todo0">ToDo 컴포넌트 만들기</lable>
        </div>
    );
  }
}

export default ToDo;
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일을 수정

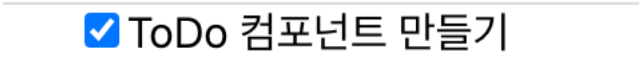
```
import React from "react";
import ToDo from "./ToDo";
import './App.css';

function App() {
  return (
    <div className="App">
      <ToDo/>
    </div>
  );
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 브라우저 확인



Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – Props 와 State

```
import React from "react";

class ToDo extends React.Component {
    constructor(props) {
        //props 오브젝트 초기화
        super(props);
        this.state = {item:props.item};
    }
```

Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – Props 와 State

```
render() {
    return (
        <div className="ToDo">
            <input
                type="checkbox"
                id={this.state.item.id}
                name={this.state.item.id}
                checked={this.state.item.done}/>
            <label id={this.state.item.id}>{this.state.item.title}</label>
        </div>
    );
}
}

export default ToDo;
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – Props 와 State

```
import React from "react";
import ToDo from "./ToDo";
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {item: { id: 0, title: "Hello React", done: true}};
  }

  render() {
    return (
      <div className="App">
        <ToDo item={this.state.item} />
      </div>
    );
  }
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 브라우저 확인 – Props 와 State

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – ToDo 리스트

```
import React from "react";
import ToDo from "./ToDo";
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
  }

  render() {
    var todoItems = this.state.items.map((item, idx) => (
      <ToDo item={item} key={item.id} />
    ));
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – ToDo 리스트

```
        return (
          <div className="App">
            {todoItems}
          </div>
        );
      }
    }

    export default App;
```

Front End

❖ ToDo 목록 보기

✓ 브라우저 확인 – ToDo 리스트

Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – material ui를 이용한 디자인

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
} from "@material-ui/core";

class ToDo extends React.Component {
    constructor(props) {
        //props 오브젝트 초기화
        super(props);
        this.state = { item: props.item };
    }
```

Front End

- ToDo 목록 보기
  - ✓ ToDo.jsx 파일 수정 – material ui를 이용한 디자인

```
render() {
    const item = this.state.item;
    return (
        <ListItem>
            <Checkbox checked={item.done} />
            <ListItemText>
                <InputBase
                    inputProps={{ "aria-label": "naked" }}
                    type="text"
                    id={item.id}
                    name={item.id}
                    value={item.title}
                    multiline={true}
                    fullWidth={true}
                />
            </ListItemText>
        </ListItem>
    );
  }
}
export default ToDo;
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – material ui를 이용한 디자인

```
import React from "react";
import ToDo from "./ToDo";
import {Paper, List} from "@material-ui/core"
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
  }
```

Front End

- ToDo 목록 보기
  - ✓ App.js 파일 수정 – material ui를 이용한 디자인

```
render() {
  var todoItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} />
        ))}
      </List>
    </Paper>
  );

  return (
    <div className="App">
      {todoItems}
    </div>
  );
 }
}

export default App;
```

Front End

- ToDo 목록 보기
  - ✓ 웹 브라우저에서 확인 – material ui를 이용한 디자인

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일을 추가하고 작성

```
import React from "react"
import { TextField, Paper, Button, Grid } from "@material-ui/core";

class AddTodo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { item: { title: "" } };

  }
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일을 추가하고 작성

```
render() {
  return (
    <Paper style={{ margin: 16, padding: 16 }}>
      <Grid container>
        <Grid xs={11} md={11} item style={{ paddingRight: 16 }}>
          <TextField
            placeholder="Add Todo here"
            fullWidth
          />
        </Grid>
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일을 추가하고 작성

```
            <Grid xs={1} md={1} item>
              <Button
                fullWidth
                color="secondary"
                variant="outlined"
              >
                +
              </Button>
            </Grid>
          </Grid>
        </Paper>
      );
    }
  }

  export default AddTodo;
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정

```
import React from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import {Paper, List, Container} from "@material-ui/core"
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done: true}]};
  }
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정

```
render() {
 var todoItems = this.state.items.length > 0 && (
   <Paper style={{ margin: 16 }}>
     <List>
       {this.state.items.map((item, idx) => (
         <ToDo item={item} key={item.id} />
       ))}
     </List>
   </Paper>
 );
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정

```
        return (
          <div className="App">
            <Container maxWidth="md">
              <AddToDo/>
              <div className="ToDoList">{todoItems}</div>
            </Container>
          </div>
        );
      }
    }

    export default App;
```

Front End

- ToDo 추가
  - ✓ 브라우저 확인

Front End

- ToDo 추가
  - ✓ App.js 파일 수정 – 이벤트 처리

```
import React from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import {Paper, List, Container} from "@material-ui/core"
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done:
true}]};
  }

  //데이터 추가를 위한 함수
  add = (item) => {
    const thisItems = this.state.items;
    item.id = "ID-" + thisItems.length; // key를 위한 id추가
    item.done = false; // done 초기화
    thisItems.push(item); // 배열에 아이템 추가
    this.setState({ items: thisItems }); // 업데이트는 반드시 this.setState로 해야됨.
    console.log("items : ", this.state.items);
  };
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정 – 이벤트 처리

```
render() {
  var todoItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id}/>
        ))}
      </List>
    </Paper>
  );
```

Front End

- ToDo 추가
  - ✓ App.js 파일 수정 – 이벤트 처리

```
        return (
          <div className="App">
            <Container maxWidth="md">
              <AddToDo add={this.add}/>
              <div className="ToDoList">{todoItems}</div>
            </Container>
          </div>
        );
      }
    }

    export default App;
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
import React from "react"
import { TextField, Paper, Button, Grid } from "@material-ui/core";

class AddTodo extends React.Component {
    constructor(props) {
        super(props);
        this.state = { item: { title: "" } };
        this.add = props.add;
    }
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
//Input 의 내용이 변경될 때 호출
onInputChange = (e) => {
    const thisItem = this.state.item;
    thisItem.title = e.target.value;
    this.setState({ item: thisItem });
    console.log(thisItem);
};

//+버튼을 눌렀을 때 호출
onButtonClick = () => {
    this.add(this.state.item);
    this.setState({ item: { title: "" } });
};

//Enter를 눌렀을 때 처리
enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        this.onButtonClick();
    }
};
```

Front End

- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
render() {
    return (
        <Paper style={{ margin: 16, padding: 16 }}>
            <Grid container>
                <Grid xs={11} md={11} item style={{ paddingRight: 16 }}>
                    <TextField
                        placeholder="Add Todo here"
                        fullWidth
                        onChange={this.onInputChange}
                        value={this.state.item.title}
                        onKeyPress={this.enterKeyEventHandler}
                    />
                </Grid>
```

Front End

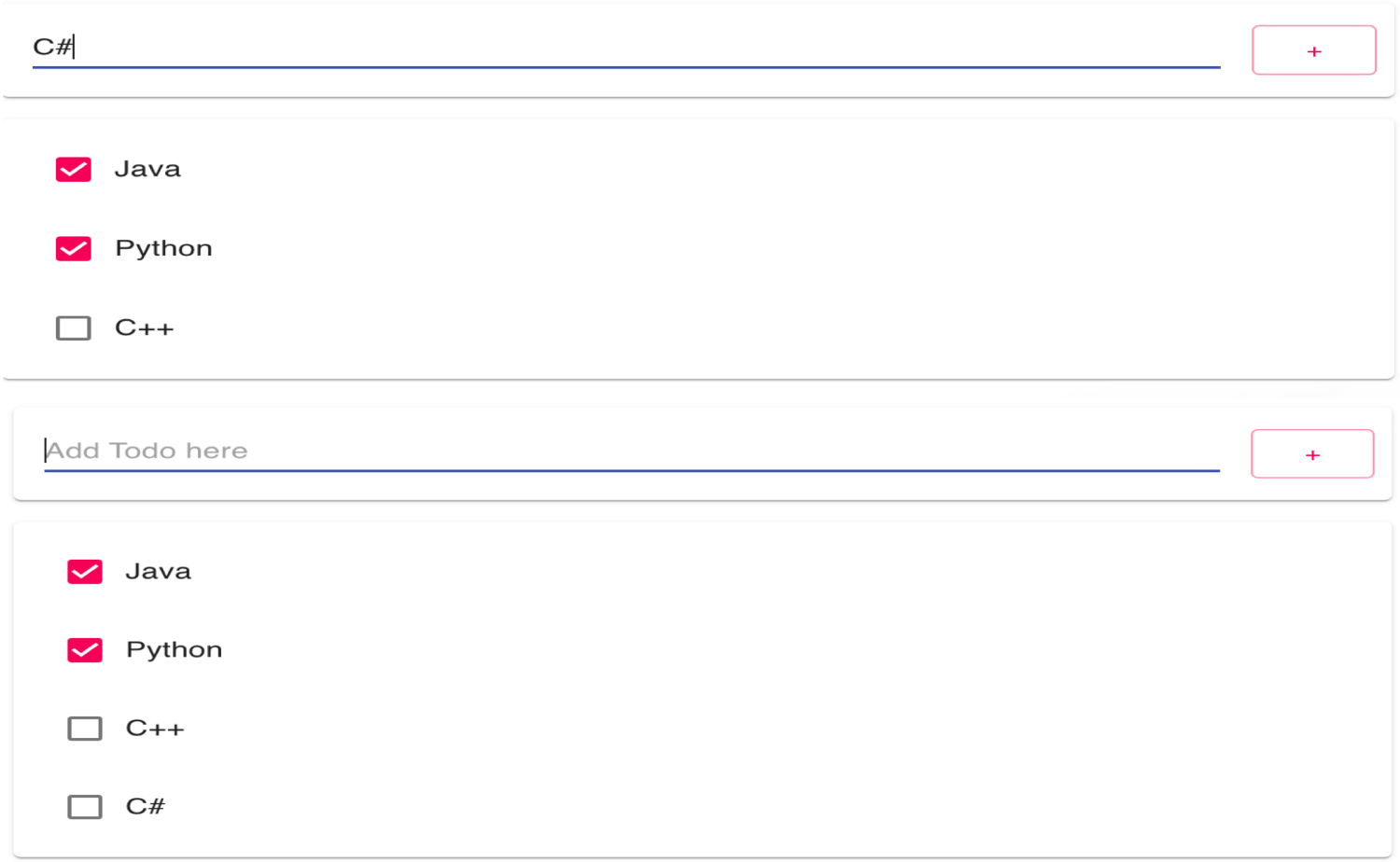
- ToDo 추가
  - ✓ AddToDo.jsx 파일 수정 – 이벤트 처리

```
                    <Grid xs={1} md={1} item>
                        <Button
                            fullWidth
                            color="secondary"
                            variant="outlined"
                            onClick={this.onButtonClick}
                        >
                            +
                        </Button>
                    </Grid>
                </Grid>
            </Paper>
        );
    }
}

export default AddTodo;
```

Front End

- ToDo 추가
  - ✓ 브라우저 확인 – 이벤트 처리



Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
class ToDo extends React.Component {
    constructor(props) {
        //props 오브젝트 초기화
        super(props);
        this.state = { item: props.item };
    }
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

```
render() {
    const item = this.state.item;
    return (
        <ListItem>
            <Checkbox checked={item.done} />
            <ListItemText>
                <InputBase
                    inputProps={{ "aria-label": "naked" }}
                    type="text"
                    id={item.id}
                    name={item.id}
                    value={item.title}
                    multiline={true}
                    fullWidth={true}
                />
            </ListItemText>
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 아이콘 출력

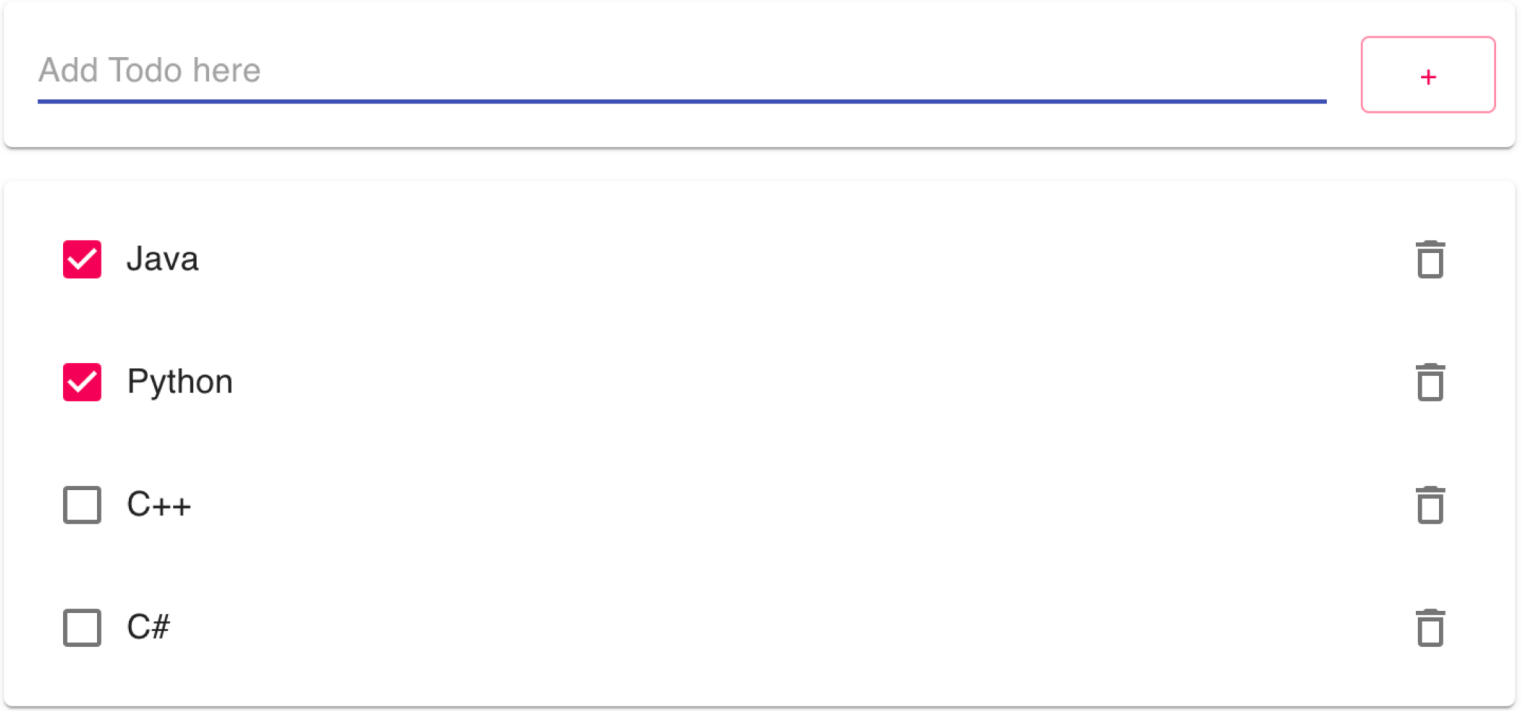
```
                    <ListItemSecondaryAction>
                        <IconButton aria-label="Delete Todo">
                            <DeleteOutlined />
                        </IconButton>
                    </ListItemSecondaryAction>
                </ListItem>
            );
        }
    }

    export default ToDo;
```

Front End

❖ ToDo 삭제

✓ 브라우저에서 확인 – 삭제 아이콘 출력



Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
import React from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import {Paper, List, Container} from "@material-ui/core"
import './App.css';

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [{ id: 0, title: "Java", done: true}, { id: 1, title: "Python", done:
true}]};
  }

  //데이터 추가를 위한 함수
  add = (item) => {
    const thisItems = this.state.items;
    item.id = "ID-" + thisItems.length; // key를 위한 id추가
    item.done = false; // done 초기화
    thisItems.push(item); // 배열에 아이템 추가
    this.setState({ items: thisItems }); // 업데이트는 반드시 this.setState로 해야됨.
    console.log("items : ", this.state.items);
  };
```

Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
delete = (item) => {
  const thisItems = this.state.items;
  console.log("Before Update Items : ", this.state.items);
  const newItems = thisItems.filter((e) => e.id !== item.id); // 해당 id 걸러내기
  this.setState({ items: newItems }, () => {
    // 디버깅 콜백
    console.log("Update Items : ", this.state.items);
  });
};
```

Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
render() {
  var todoItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id} delete={this.delete}/>
        ))}
      </List>
    </Paper>
  );
```

Front End

- ToDo 삭제
  - ✓ App.js 파일 수정 – 삭제 구현

```
        return (
          <div className="App">
            <Container maxWidth="md">
              <AddToDo add={this.add}/>
              <div className="ToDoList">{todoItems}</div>
            </Container>
          </div>
        );
      }
    }

    export default App;
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
class ToDo extends React.Component {
    constructor(props) {
        //props 오브젝트 초기화
        super(props);
        this.state = { item: props.item };
        this.delete = props.delete;
    }

    deleteEventHandler = () => {
        this.delete(this.state.item);
    };
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
render() {
    const item = this.state.item;
    return (
        <ListItem>
            <Checkbox checked={item.done} />
            <ListItemText>
                <InputBase
                    inputProps={{ "aria-label": "naked" }}
                    type="text"
                    id={item.id}
                    name={item.id}
                    value={item.title}
                    multiline={true}
                    fullWidth={true}
                />
            </ListItemText>
```

Front End

- ToDo 삭제
  - ✓ ToDo.jsx 파일 수정 – 삭제 구현

```
                    <ListItemSecondaryAction>
                        <IconButton aria-label="Delete Todo"
onClick={this.deleteEventHandler}>
                            <DeleteOutlined />
                        </IconButton>
                    </ListItemSecondaryAction>
                </ListItem>
            );
        }
}

export default ToDo;
```

Front End

❖ ToDo 삭제

✓ 웹 브라우저에서 확인 – 삭제 구현

Front End

- ToDo 수정
  - ✓ 요구 사항
    - Todo 컴포넌트에 readonly 플래그가 있어 readonly가 true인 경우 아이템 수정이 불가능하고 false인 경우 아이템을 수정할 수 있음
    - 사용자가 어떤 아이템의 title을 클릭하면 해당 input field는 수정할 수 있는 상태인 readonly를 false인 상태로 변경
    - 사용자가 Enter키 또는 Retrun키를 누르면 readonly가 true인 상태로 전환됨
    - 체크박스 클릭 시 item.done 값을 전환

Front End

❖ ToDo 수정

✓ ToDo.jsx 수정

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";

class ToDo extends React.Component {
    constructor(props) {
        super(props);
        this.state = { item: props.item, readOnly: true };
        this.delete = props.delete;
    }
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
deleteEventHandler = () => {
    this.delete(this.state.item);
};

offReadOnlyMode = () => {
    console.log("Event!", this.state.readOnly);
    this.setState({ readOnly: false }, () => {
        console.log("ReadOnly? ", this.state.readOnly);
    });
};

enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        this.setState({ readOnly: true });
    }
};
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
editEventHandler = (e) => {
    const thisItem = this.state.item;
    thisItem.title = e.target.value;
    this.setState({ item: thisItem });
};

checkboxEventHandler = (e) => {
    const thisItem = this.state.item;
    thisItem.done = !thisItem.done;
    this.setState({ item: thisItem });
};
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
render() {
    const item = this.state.item;
    return (
        <ListItem>
            <Checkbox checked={item.done} onChange={this.checkboxEventHandler}
/>
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
<ListItemText>
    <InputBase
        inputProps={{
            "aria-label": "naked",
            readOnly: this.state.readOnly,
        }}
        type="text"
        id={item.id}
        name={item.id}
        value={item.title}
        fullWidth={true}
        onClick={this.offReadOnlyMode}
        onChange={this.editEventHandler}
        onKeyPress={this.enterKeyEventHandler}
    />
</ListItemText>
```

Front End

- ToDo 수정
  - ✓ ToDo.jsx 수정

```
                    <ListItemSecondaryAction>
                        <IconButton
                            aria-label="Delete Todo"
                            onClick={this.deleteEventHandler}
                        >
                            <DeleteOutlined />
                        </IconButton>
                    </ListItemSecondaryAction>
                </ListItem>
            );
        }
    }

    export default ToDo;
```

Front End

❖ ToDo 수정

✓ 웹 브라우저에서 확인

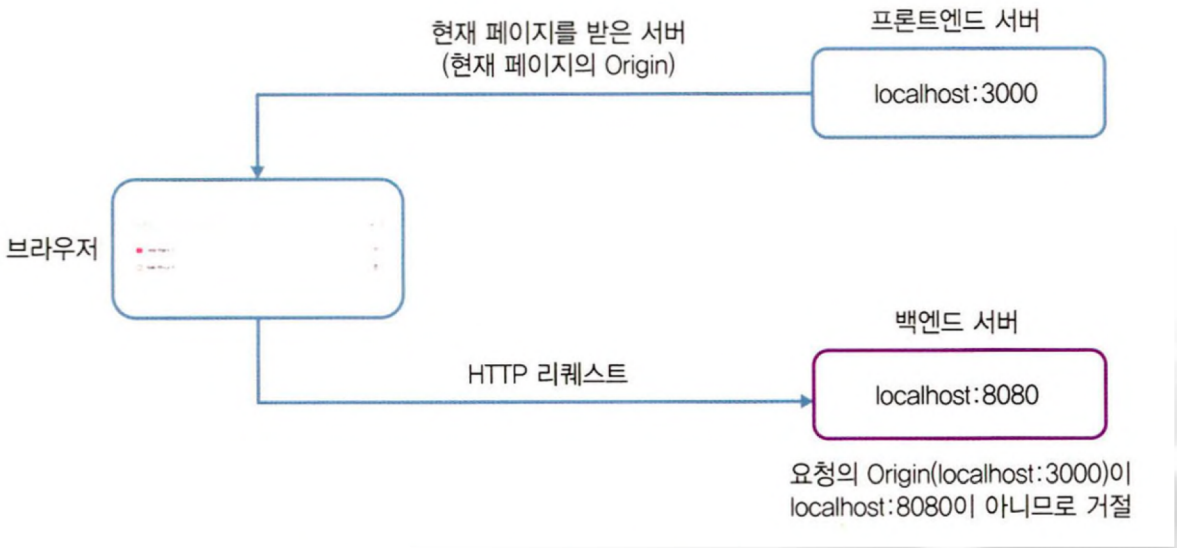
Front End

- 데이터 초기화
  - ✓ App.js 파일 수정 – 기본 데이터 삭제

```
class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: [ ]};
  }
```

CORS

- CORS
  - ✓ Cross-Origin Resource Sharing 의 약자(https://en.wikipe dia.org/wiki/Cross-origin_resource_sharing)로 처음 리소스를 제공한 도메인(Origin)이 현재 요청하려는 도메인과 다르더라도 요청을 허락해 주는 웹 보안 방침



CORS

- CORS
  - ✓ FrontEnd Project 의 App.js 파일의 컴포넌트에 추가

```
componentDidMount() {
  const requestoptions = {
    method: "GET",
    headers: { "Content-Type": "application/json" },
  };

  fetch("http://localhost/todo", requestoptions)
    .then((response) => response.json())
    .then((response) => {
      this.setState({
        items: response.list
      });
  },
    (error) => {
      console.log(error)
      this.setState({
        error
      });
  }
  );
  }
```

CORS

- CORS
  - ✓ 브라우저의 검사 창에서 확인 – 에러 발생

```
Access to fetch at 'http://localhost/todo' from origin 'http://localhost:300 localhost/:1
0' has been blocked by CORS policy: Response to preflight request doesn't pass access
control check: No 'Access-Control-Allow-Origin' header is present on the requested
resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to
fetch the resource with CORS disabled.
Failed to load resource: net::ERR_FAILED                                    /todo:1
```

CORS

- Django 에서의 CORS 설정
  - ✓ 패키지 설치
    - pip install django-cors-headers

CORS

- Django 에서의 CORS 설정
  - ✓ settings.py 파일 수정
    - 애플리케이션 등록

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "rest_framework",
    "todoapplication",
    "corsheaders"
]
```

CORS

- Django 에서의 CORS 설정
  - ✓ settings.py 파일 수정
    - 미들웨어 등록

```
MIDDLEWARE = [
    "corsheaders.middleware.CorsMiddleware",
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

CORS

- Django 에서의 CORS 설정
  - ✓ settings.py 파일 수정
    - ● 허용할 List 설정

```
CORS_ORIGIN_WHITELIST = ['http://127.0.0.1:3000', 'http://localhost:3000']
CORS_ALLOW_CREDENTIALS = True
```

Front End

❖ Back-End 의 URL을 저장할 환경 설정 파일을 생성 – src/app-config.js

```
let backendHost;

const hostname = window && window.location && window.location.hostname;

if (hostname === "localhost") {
  backendHost = "http://localhost";
}

export const API_BASE_URL = `${backendHost}`;
```

Front End

❖ BackEnd의 데이터를 가져오는 함수를 별도의 파일에 생성 – src/service/ApiService.js

```
import { API_BASE_URL } from "../app-config";

export function call(api, method, request) {
  let options = {
    headers: new Headers({
      "Content-Type": "application/json",
    }),
    url: API_BASE_URL + api,
    method: method,
  };
  if (request) {
    // GET method
    options.body = JSON.stringify(request);
  }
```

Front End

- BackEnd의 데이터를 가져오는 함수를 별도의 파일에 생성 – src/service/ApiService.js

```
return fetch(options.url, options).then((response) =>
  response.json().then((json) => {
    if (!response.ok) {
      // response.ok가 true이면 정상적인 리스폰스를 받은것, 아니면 에러 리스폰스를 받은것.
      return Promise.reject(json);
    }
    return json;
  })
);
}
```

Front End

- App.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
import React from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import {Paper, List, Container} from "@material-ui/core"
import './App.css';
import { call } from "./service/ApiService";

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: []};
  }
```

Front End

- App.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
componentDidMount() {
    call("/todo", "GET", null).then((response) =>
      this.setState({ items: response.list })
    );
  }

  add = (item) => {
    item.userid="itstudy";
    call("/todo", "POST", item).then((response) =>
      this.setState({ items: response.list })
    );
  };

  delete = (item) => {
    call("/todo", "DELETE", item).then((response) =>
      call("/todo", "GET", null).then((response) =>
        this.setState({ items: response.list })
      )
    );
  };
```

Front End

- app.js 파일의 내용 수정 – 삽입, 삭제, 가져오기 적용

```
render() {
  var todoItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item} key={item.id} delete={this.delete}/>
        ))}
      </List>
    </Paper>
  );

  return (
    <div className="App">
      <Container maxWidth="md">
        <AddToDo add={this.add}/>
        <div className="ToDoList">{todoItems}</div>
      </Container>
    </div>
  );
 }
}

export default App;
```

Front End

❖ 브라우저 와 데이터베이스에서 확인

123 done	ABC title	ABC user_id
0	Java	temporary-user
0	Python	temporary-user

Front End

- app.js 파일의 내용 수정 – 수정 적용

```
import React from "react";
import ToDo from "./ToDo";
import AddToDo from "./AddToDo";
import {Paper, List, Container} from "@material-ui/core"
import './App.css';
import { call } from "./service/ApiService";

class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {items: []};
  }

  componentDidMount() {
    call("/todo", "GET", null).then((response) =>
      this.setState({ items: response.data })
    );
  }
```

Front End

- app.js 파일의 내용 수정 – 수정 적용

```
add = (item) => {
  call("/todo", "POST", item).then((response) =>
    this.setState({ items: response.data })
  );
};

delete = (item) => {
  call("/todo", "DELETE", item).then((response) =>
    this.setState({ items: response.data })
  );
};

update = (item) => {
  call("/todo", "PUT", item).then((response) =>
    call("/todo", "GET", null).then((response) =>
      this.setState({ items: response.list })
    )
  );
};
```

Front End

❖ app.js 파일의 내용 수정 – 수정 적용

```
render() {
  var todoItems = this.state.items.length > 0 && (
    <Paper style={{ margin: 16 }}>
      <List>
        {this.state.items.map((item, idx) => (
          <ToDo item={item}
          key={item.id}
          delete={this.delete}
          update={this.update}/>
        ))}
      </List>
    </Paper>
  );
```

Front End

- app.js 파일의 내용 수정 – 수정 적용

```
        return (
          <div className="App">
            <Container maxWidth="md">
              <AddToDo add={this.add}/>
              <div className="ToDoList">{todoItems}</div>
            </Container>
          </div>
        );
      }
    }

    export default App;
```

Front End

❖ ToDo.jsx 파일의 내용 수정 – 수정 적용

```
import React from "react";

import {
    ListItem,
    ListItemText,
    InputBase,
    Checkbox,
    ListItemSecondaryAction,
    IconButton
} from "@material-ui/core";

import DeleteOutlined from "@material-ui/icons/DeleteOutlined";

class ToDo extends React.Component {
    constructor(props) {
        super(props);
        this.state = { item: props.item, readOnly: true };
        this.delete = props.delete;
        this.update = props.update;
    }
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
deleteEventHandler = () => {
    this.delete(this.state.item);
};

offReadOnlyMode = () => {
    console.log("Event!", this.state.readOnly);
    this.setState({ readOnly: false }, () => {
        console.log("ReadOnly? ", this.state.readOnly);
    });
};

enterKeyEventHandler = (e) => {
    if (e.key === "Enter") {
        this.setState({ readOnly: true });
        this.update(this.state.item);
    }
};
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
editEventHandler = (e) => {
    const thisItem = this.state.item;
    thisItem.title = e.target.value;
    this.setState({ item: thisItem });
};

checkboxEventHandler = (e) => {
    const thisItem = this.state.item;
    thisItem.done = !thisItem.done;
    this.setState({ item: thisItem });
    this.update(this.state.item);
};
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
render() {
    const item = this.state.item;
    return (
        <ListItem>
            <Checkbox checked={item.done} onChange={this.checkboxEventHandler} />
            <ListItemText>
                <InputBase
                    inputProps={{
                        "aria-label": "naked",
                        readOnly: this.state.readOnly,
                    }}
                    type="text"
                    id={item.id}
                    name={item.id}
                    value={item.title}
                    fullWidth={true}
                    onClick={this.offReadOnlyMode}
                    onChange={this.editEventHandler}
                    onKeyPress={this.enterKeyEventHandler}
                />
            </ListItemText>
```

Front End

- ToDo.jsx 파일의 내용 수정 – 수정 적용

```
                    <ListItemSecondaryAction>
                        <IconButton
                            aria-label="Delete Todo"
                            onClick={this.deleteEventHandler}
                        >
                            <DeleteOutlined />
                        </IconButton>
                    </ListItemSecondaryAction>
                </ListItem>
            );
        }
    }

    export default ToDo;
```