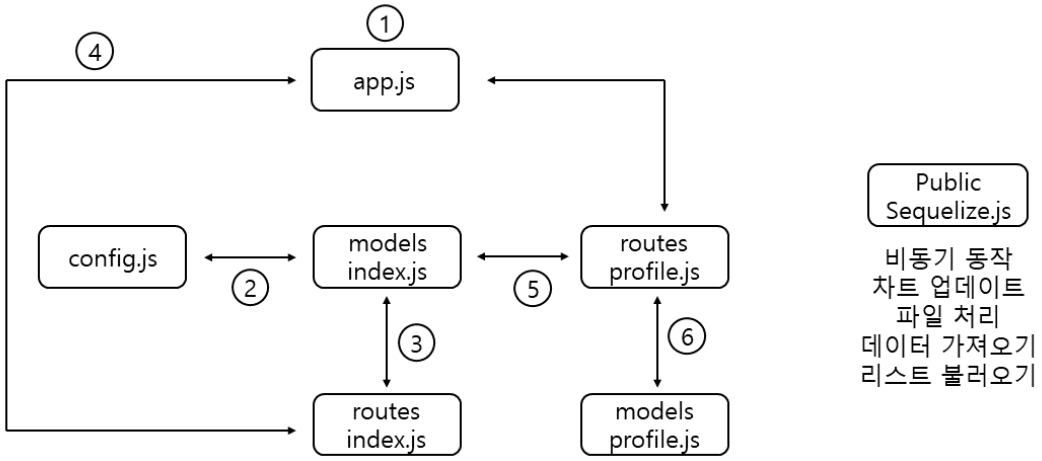
웹 응용기술 보고서

Node.js를 활용한 Profiler 프로그램 분석

학과	컴퓨터공학과
학번	20200831
이름	나성훈

목 차

1. 프로그램 수행 절차 분석



1. app.js를 사용하여 서버를 구동시킨다.
    1.1. 기본페이지 GET 요청 시
        2. config.js의 DB사용자 정보를 가져와 models index.js에서 DB의 연결한다.
        3. routes index.js에서 models index.js의 getTableList()를 가져온다.
        4. main페이지 접속 시 routes index.js를 이용해 가져온 테이블 리스트를 render한다.

    1.2. 기본 외 페이지 요청 시
        5. routes profile.js에서 models index.js의 기능을 가져온다.
        6. routes profile.js
            6.1 POST / 요청 시 getTableList()의 쿼리문을 이용해 테이블의 데이터를 가져온다.
            6.2 GET /data/:tableName 요청 시 getTableList()를 이용해 데이터를 가져온 후 DB와 init을 통해 연결한다. 이 후 각 정보를 json형식으로 반환한다.
            6.3 DELETE /drop/:tableName 요청 시 droptTable()의 쿼리문을 이용해 테이블을 삭제한다.
            6.4 GET /coredata/:tableName/:core 요청 시 init을 통해 DB와 연결한다. 이 후 해당 코어에 태스크 정보를 처리한 뒤 json형식으로 반환한다.
            6.5 GET /taskdata/:tableName/:task 요청 시 init을 통해 DB와 연결한다. 이 후 해당 태스크에 코어 정보를 처리한 뒤 json형식으로 반환한다.

파일명	파일 정보
app.js	express를 이용해 서버를 열고 넌적스를 사용한다.
config.js	DB사용자 정보에 대한 데이터 저장 파일이다.
models_index.js	DB의 기본 기능이 정의된 파일이다.
models_profile.js	DB테이블에 대한 기본 정보를 담은 파일이다.
routes_index.js	/페이지 진입시 테이블을 가져오는 파일이다.
routes_profile.js	페이지 내에서 특정 조건시 해당 라우터를 실행하는 파일이다.
views	HTML로 페이지를 구성하는 파일이다.

2. 기존 소스 코드 분석

config.json

데이터베이스 연결에 필요한 정보를 입력한다.

```
{
    "development": {
      "username": "root",
      "password": "h2640495",
      "database": "javaweb",
      "host": "127.0.0.1",
      "dialect": "mysql",
      "logging": false
    }
}
```

models => index.js

데이터베이스 연결 후 쿼리 사용

```
const Sequelize = require('sequelize'); //시퀄라이즈 모듈 추가

const env = process.env.NODE_ENV || 'development';
const config = require('../config/config')[env]; //config파일 불러오기
const db = {};

const sequelize = new Sequelize(config.database, config.username,
config.password, config); //데이터베이스 연결


async function createTable(tableName){ //테이블 생성
    const Model = sequelize.define( //모델 생성
        tableName,
        {
            core : { //코어(속성)
                type: Sequelize.STRING(20), //가변문자열 20
                allowNull : false, //널 허용 안함
            },
            task : { //태스크(속성)
                type: Sequelize.STRING(20), //가변문자열 20
                allowNull : false, //널 허용 안함
            },
            usaged : { //사용량(속성)
                type:Sequelize.INTEGER.UNSIGNED, //unsigned int형
                allowNull:false, //널 허용 안함
            },
```

```
        },
        {
            sequelize,
            timestamps: false, //자동 타임스탬프 사용 안함
            underscored: false, //자동으로 스네이크 케이스로 변환 안함
            modelName: 'Profile', //모델 이름 = Profile
            tableName: tableName, //테이블 이름 설정
            paranoid: false, //소프트삭제 비허용 => 데이터를 실제로 삭제
            charset: 'utf8', //한글 사용
            collate: 'utf8_general_ci', //정렬방식 AaBb순으로 정렬
        }
    );

    await Model.sync(); //테이블 생성 => 이미 있을 경우 생성 안함

    return Model;
}

async function dropTable(tableName) { //테이블 삭제
    try {
        await sequelize.query(`DROP TABLE IF EXISTS \`${tableName}\``); //테이블 삭제
        console.log(`테이블 '${tableName}'이(가) 삭제되었습니다.`); //콘솔 출력
    } catch (error) { //에러 발생 시
      console.error(`테이블 삭제 중 오류가 발생했습니다: ${error}`); //에러 출력
    }
}

async function createDynamicTable(profile){ //동적 테이블 생성
    // console.log(profile);
    const tableName = profile[0][0]; //프로파일 0,0 요소 이름으로 설정
    const DynamicModel = await createTable(tableName); //테이블 생성해서 동적 테이블로 저장

    let core_row = -1; //변수 초기화
    for(let row = 1; row<profile.length; row++){ //프로파일 각 행 반복
        if(core_row == -1){ //현재 행이 코어 행이면 다음 행으로 진행
            core_row = row;
            continue;
        }
```

```
            if(profile[row].length==1){ //현재 행 길이가 1이면 코어행 리셋하고 다음 행
                core_row = -1;
                continue;
            }
            for(let column = 1; column < profile[row].length; column++){ //각 열 반복
                try{
                await DynamicModel.create({ //DynamicModel에 데이터 삽입
                    task: profile[core_row][column-1], //코어 행 (열-1)
                    core : profile[row][0], //현재 행 첫 번째 값
                    usaged : profile[row][column], //현재 행 현재 열 값
                });
                }catch(e){ //오류 발생 시
                    console.log(`Error: ${tableName} 파일 데이터 오류 발생`);//오류 출력
                }
            }
        }
}

async function getTableList() { //테이블리스트 불러오기
    const query = 'SHOW TABLES'; // 데이터베이스별로 조회 방식이 다를 수 있으므로 사용하는 데이터베이스에 맞는 쿼리를 사용

    // 쿼리 실행
    const [results, metadata] = await sequelize.query(query);

    // 테이블 목록 추출
    const tableList = results.map((result) => result.Tables_in_javaweb); // 'database'는 실제 사용하는 데이터베이스 이름으로 변경

    return tableList;
}

db.sequelize = sequelize;

module.exports = {db, createDynamicTable, sequelize, getTableList, dropTable};
```

models => profile.js

DB에서 불러올 때 가지고 올 데이터에 속성 및 제약사항 정의

```
const Sequelize = require('sequelize');

class Profile extends Sequelize.Model{ //시퀄라이즈 모델
    static initiate(sequelize,tableName){ //테이블 초기 세팅, DB연결 과정
        Profile.init({
            core : { //코어
                type: Sequelize.STRING(20), //가변문자열20
                allowNull : false, //널값 허용 안함
            },
            task : { //태스크
                type: Sequelize.STRING(20), //가변문자열20
                allowNull : false, //널값 허용 안함
            },
            usaged : { //사용값
                type:Sequelize.INTEGER.UNSIGNED, //unsigned int
                allowNull:false, //널값 허용 안함
            },
        },
        {
            sequelize,
            timestamps: false, //타임스탬프 자동 생성 안함
            underscored: false, //자동으로 스네이크 케이스로 변환 안함
            modelName: 'Profile', //모델이름 = Profile
            tableName: tableName, //테이블 이름 지정
            paranoid: false, //소프트삭제 비허용 => 데이터를 실제로 삭제
            charset: 'utf8', //한글 사용
            collate: 'utf8_general_ci', //정렬방식 AaBb순으로 정렬
        });
    }

    static associations(db){ //db 관계설정

    }
};

module.exports = Profile; //모듈 활성화
```

public => sequelize.js

```
let fileName="";
let selete="";

const profileList = document.querySelectorAll('#profile_list tr td:first-child'); //첫 번째 profileList에 td(셀) 모두 선택
profileList.forEach((el) => { //각 셀에 클릭 이벤트 리스너 추가
    el.addEventListener('click', function () {
        fileName = el.textContent; //클릭한 셀 fileName에 저장
        profileList.forEach((otherEl) => { //모든 셀 배경색 = 흰색
            otherEl.style.setProperty("background-color", "white");
        });
        this.style.setProperty("background-color", "#888888");//클릭한 셀 배경색 = 회색
        select = undefined;
        if (chart) { //chart가 존재하면 삭제
            chart.destroy();
        }
        getdata(); //getdata호출
    });
});

document.getElementById('profile_form').addEventListener('submit', async (e) => { //profile_form 제출 누르면
    e.preventDefault(); //비동기 활성화를 위해 기본 폼 제출 방지

    const files = document.querySelector('#input_profile').files; //input_profile가져오기
    let profiles = [];
    let is_error = false;
    if(!files){ //files이 등록되지 않으면
        return alert('파일을 등록하세요'); //파일 등록 알람 표시
    }
    const filePromises = Array.from(files).map((file) => { //파일 -> 배열로 변환
        if (file.name.split(".").pop().toLowerCase() === 'txt') { //.txt파일이면
            return new Promise((resolve, reject) => {
                readTextFile(file, (data) => { //읽어드림
                    profiles.push(data); //프로파일에 데이터 저장
                    resolve(); //프로미스 성공반환
                });
            });
```

```
        } else { //.txt파일이 아니면
            alert(".txt파일만 입력해주세요"); //.txt파일 요청 알람 표시
            is_error = true; //에러 활성화
        }
    });

    await Promise.all(filePromises);

    if(!is_error){ //에러가 아니면
        const response = await fetch('/profiles',{
            method: 'POST', //POST형식으로 요청
            headers: {
                'Content-Type' : 'application/json' //JSON 파일 요청
            },
            body: JSON.stringify(profiles) //profiles배열 문자열로 변환
        });

        if (response.ok) { //응답 성공시
            response.json().then(data => { //json parse하는 과정(비동기 사용을 위해
JSON.parse()대신 사용)
                getList(); //리스트를 가져옴
                alert(data.message); //data 메시지 알람 표시
            });
        } else { //응답 실패시
            console.error('파일 전송 중 오류가 발생하였습니다.'); //오류 출력
        }
    }
});

async function getList(){
    const res = await axios.get('profiles'); //AJAX사용 해 profiles 요청
    const profiles = res.data; //비동기 방식으로 응답받은 데이터

    const tbody = document.querySelector('#profile_list tbody'); //profile_list아이
디를 가진 tvody객체 생성
    tbody.innerHTML = ''; //tbody 내용 초기화
    profiles.map(function(profile){
        const row = document.createElement('tr'); //새 행 생성
        const td = document.createElement('td'); //새 셀 생성
        td.textContent = profile; //td에 프로필 이름 삽입
```

```
        td.className= 'text-center fw-semibold'; //td에 클래스 추가
        td.addEventListener('click',function(){ //td 클릭시
            fileName = profile; //클릭된 프로필 filename에 저장
            const profileList = document.querySelectorAll('#profile_list tr td:first-child'); //모든 첫 번째 셀 선택
            profileList.forEach((otherEl) => {
                otherEl.style.setProperty("background-color", "white"); //다른 셀 배경색 흰색으로 변경
            });
            this.style.setProperty("background-color", "#888888"); //클릭된 셀 배경색 회색으로 변경
            if (chart) { //차트가 존재하면
                chart.destroy(); //차트 삭제
            }
            getdata(); //데이터 가져오기
        });
        if(profile==fileName) td.style.setProperty("background-color", "#888888"); //현재 선택된 프로필이면 배경색 회색으로 변경
        row.appendChild(td); //행에 셀 추가
        const td2 = document.createElement('td'); //두 번째 셀 생성
        const btndrop = document.createElement('button'); //삭제 버튼 생성
        btndrop.textContent="삭제"; //버튼 텍스트 설정
        btndrop.className="btn btn-danger"; //버튼 클래스 추가
        btndrop.addEventListener('click',function(){ deleteTable(`${profile}`); }); //버튼 클릭시 클릭된 profile제거
        td2.appendChild(btndrop); //td2에 삭제 버튼 추가
        row.appendChild(td2); //행에 td2 추가

        tbody.appendChild(row); //tbody에 행 추가
    });
}

async function deleteTable(name){ //비동기 함수 테이블 삭제
    await axios.delete(`profiles/drop/${name}`); //해당 이름의 profiles 삭제
    if(fileName==name && chart) { //현재 선택된 파일 이름과 차트가 같을 시
        chart.destroy(); //차트 삭제
        const task_div = document.querySelector('#core'); //코어 해당 요소 삽입
        task_div.innerHTML=""; //선택된 요소 제거
        const core_div = document.querySelector('#task'); //태스크 해당 요소 삽입
        core_div.innerHTML = ''; //선택된 요소 제거
```

```
        fileName = ""; //파일이름 초기화
    };
    setTimeout(getList,50); //50ms후 getList 호출
}

async function getdata(){ //데이터 가져오기

    const res = await axios.get(`profiles/data/${fileName}`);
    const cores = res.data.cores; //응답에서 cores데이터 가져오기
    const tasks = res.data.tasks; //응답에서 tasks데이터 가져오기

    const task_div = document.querySelector('#core'); //id core인 요소 가져오기
    task_div.innerHTML = 'select core : '; //요소 slect core : 로 설정
    tasks.map(function(task){ //task배열 순회
        let button = document.createElement('button'); //버튼 생성
        button.className = 'btn btn-info me-2'; //버튼 클래스 추가
        button.textContent = task.core; //버튼 이름 task.core로 설정
        button.addEventListener('click', function(){ //버튼 클릭 시
            updateChart('task', task.core); //차트 업데이트
            const coreDiv = document.getElementById('core'); //id core 요소 가져
오기
            const coreBtns = coreDiv.getElementsByClassName('btn'); //coreDiv에
btn 요소 가져오기
            for (let i = 0; i < coreBtns.length; i++) { //core모든 버튼
                coreBtns[i].className = "btn btn-info me-2"; //클래스 추가
            }
            const taskDiv = document.getElementById('task'); //id task 요소 가져
오기
            const taskBtns = taskDiv.getElementsByClassName('btn'); //taskDiv에
btn 요소 가져오기
            for (let i = 0; i < taskBtns.length; i++) { //task모든 버튼
                taskBtns[i].className = "btn btn-success me-2"; //클래스 추가
            }
            this.className = "btn btn-secondary me-2"; //클릭된 버튼 클래스 추가
        });
        task_div.appendChild(button); //생성된 버튼 task_div 자식으로 추가
    });

    const core_div = document.querySelector('#task'); //id task 요소 가져오기
    core_div.innerHTML = 'select task : '; //요소 slect task : 로 설정
```

```
    cores.map(function(core){ //cores 배열 순회
        let button = document.createElement('button'); //버튼 생성
        button.className = 'btn btn-success me-2'; //클래스 추가
        button.textContent = core.task; //버튼 이름 core.task로 설정
        button.addEventListener('click', function(){ //버튼 클릭 시
            updateChart('core', core.task); //차트 업데이트
            const coreDiv = document.getElementById('core'); //id core 요소 가져
오기
            const coreBtns = coreDiv.getElementsByClassName('btn'); //coreDiv에
btn 요소 가져오기
            for (let i = 0; i < coreBtns.length; i++) { //모든 코어 버튼
                coreBtns[i].className = "btn btn-info me-2"; //클래스 추가
            }
            const taskDiv = document.getElementById('task'); //id task 요소 가져
오기
            const taskBtns = taskDiv.getElementsByClassName('btn'); //taskDiv에
btn 요소 가져오기
            for (let i = 0; i < taskBtns.length; i++) { //모든 태스크 버튼
                taskBtns[i].className = "btn btn-success me-2"; //클래스 추가
            }
            this.className = "btn btn-secondary me-2"; //클릭된 버튼 클래스 추가
        });
        core_div.appendChild(button); //생성된 버튼 core_div 자식으로 추가
    });

}

function readTextFile(file, save) { //파일 읽기
    const reader = new FileReader(); //객체 생성

    reader.onload = async function(event) { //파일 읽었을 때 이벤트 등록
        const contents = event.target.result; //파일 내용 가져오기
        let line_parse = contents.split("\n"); //줄 단위로 분리
        const parse = [[file.name]]; //파일 이름 첫 번째 요소로 포함한 배열 생성
        for(let i=0; i<line_parse.length; i++){ //줄 수 만큼
            parse.push(line_parse[i].trim().split(/\t| |,|\//)); //탭, 공백, 쉼표, 슬래
시 구분
        }
        save(parse); //parse에 저장
    };
```

```
    reader.onerror = function(event){ //에러 발생 시
        console.error("잘못된 파일"); //에러 출력
    }

    reader.readAsText(file, 'UTF-8'); //UTF-8 형식으로 읽음

}

let chart;
let chart_type = 'line';
let labels = [];
let minData = [];
let maxData = [];
let avgData = [];

const btnline = document.getElementById('line');
const btnbar = document.getElementById('bar');
const btnpolarArea = document.getElementById('polarArea');

btnline.addEventListener('click', function () { //line버튼 클릭 시
    chart_type = 'line'; //타입 = line
    btnline.className="btn btn-secondary"; btnbar.className="btn btn-primary";
btnpolarArea.className="btn btn-primary"; //클래스 추가
    if(fileName.length!=0) updateChart(null,null); //파일이 있는 경우 차트 업데이트
})
btnbar.addEventListener('click', function () { //bar버튼 클릭 시
    chart_type = 'bar'; //타입 = bar
    btnline.className="btn btn-primary"; btnbar.className="btn btn-secondary";
btnpolarArea.className="btn btn-primary"; //클래스 추가
    if(fileName.length!=0) updateChart(null,null); //파일이 있는 경우 차트 업데이트
});
btnpolarArea.addEventListener('click', function () { polarArea버튼 클릭 시
    chart_type = 'polarArea'; //타입 = polarArea
    btnline.className="btn btn-primary"; btnbar.className="btn btn-primary";
btnpolarArea.className="btn btn-secondary"; //클래스 추가
    if(fileName.length!=0) updateChart(null,null); //파일이 있는 경우 차트 업데이트
 });

async function updateChart(type, choose_name){ //차트 업데이트
```

```
    const profiler = document.getElementById('profiler').getContext('2d');  //id profiler인 2Dcontext 가져오기
    if (chart) { //차트가 존재하면
        chart.destroy(); //차트 삭제
    }

    if(type == 'core'){ //코어일 경우
        select = choose_name; //사용자가 선택한 파일 이름 저장
        const res = await axios.get(`profiles/taskdata/${fileName}/${select}`); //선택한 정보 요청
        const datas = res.data; //응답 데이터 저장

        labels = [];
        minData = [];
        maxData = [];
        avgData = [];
        datas.forEach((data) => {
            labels.push(data.core); //코어 이름
            minData.push(data.min_usaged); //최소 사용량 저장
            maxData.push(data.max_usaged); //최대 사용량 저장
            avgData.push(data.avg_usaged); //평균 사용량 저장
        });

    }else if(type == 'task'){ //태스크일 경우
        select = choose_name; //사용자가 선택한 파일 이름 저장
        const res = await axios.get(`profiles/coredata/${fileName}/${select}`); //선택한 정보 요청
        const datas = res.data; //응답 데이터 저장

        labels = [];
        minData = [];
        maxData = [];
        avgData = [];
        datas.forEach((data) => {
            labels.push(data.task); //태스크 이름
            minData.push(data.min_usaged); //최소 사용량 저장
            maxData.push(data.max_usaged); //최대 사용량 저장
            avgData.push(data.avg_usaged); //평균 사용량 저장
        });
```

```
    }
    if(fileName==undefined || select==undefined) return; //파일이나 선택된 항목이 없을 경우 중지

    chart = new Chart(profiler, { //차트 객체 생성
        type: `${chart_type}`, //객체 속성
        data: {
          labels: labels,
          datasets: [{
            label: 'Min', //최소
            data: minData, //최소 데이터
            borderColor: 'rgba(0, 0, 255, 0.5)', //선 색
            backgroundColor: 'rgba(0, 0, 255, 0.5)', //배경색
          }, {
            label: 'Max', //최대
            data: maxData, //최대 데이터
            borderColor: 'rgba(255, 0, 0, 1)', //선 색
            backgroundColor: 'rgba(255, 0, 0, 0.5)', //배경색
          }, {
            label: 'Avg', //평균
            data: avgData, //평균 데이터
            borderColor: 'rgba(100, 255, 30, 1)', //선 색
            backgroundColor: 'rgba(100, 255, 30, 0.5)', //배경색
          }]
        },
        options: { //옵션
          maintainAspectRatio: false, //종횡비 유지 안함
          scales: {
            y: {
              beginAtZero: true //y축 0에서 시작
            }
          },
          plugins: { //플러그인
            title: { //타이틀
              display: true, //표시 여부
              text: `${fileName}의 ${select} 정보`, //텍스트 정보
              font: {
                size: 30 //폰트 크기
              }
```

```
                    }
                }
            },
        });


}
```

routes => index.js

기본 경로와 관련된 처리

```
const express = require('express');
const router = express.Router();

const { getTableList } = require('../models/index');      //db, createDynamicTable, sequelize, getTableList, dropTable 가져옴 => 이중에서 DB 불러오기 기능만 사용

//이벤트 핸들러(테이블을 가져온다, 오류 발생시 에러 메시지 출력)
router.get('/', async (req,res)=>{
    //DB 불러오기 실행
    getTableList()
        //정상적 수행
        .then((tableList) => {
            res.render('index', {tableList});
        })

        //오류메시지
        .catch((error) => {
        console.error('테이블 리스트 조회 중 오류가 발생하였습니다:', error);
        });
});

module.exports = router;      //router(DB불러오기)기능 배포
```

routes => profile.js

특정 경로와 관련된 처리

```
const express = require('express');     //express 서버 구성
const router = express.Router();     //라우팅을 위한 객체 생성
const { createDynamicTable, getTableList, sequelize, dropTable } = require('../models/index');//models index 기능 가져오기
const profile_model = require('../models/profile');//DB 초기 설정 가져오기

// 메인 페이지에서, POST(생성)요청시 처리(input 파일 파싱하는 부분)
router.post('/', async (req, res) => {
    const profiles = req.body;      //클라이언트요청문으로 보내는 body정보 가져옴(input 파일 내용 가져오는거, 어느정도 정제된 input 파일, 3차원 배열 형태임)
    let count = 0;
    try {
        const tableList = await getTableList();     //DB 테이블 불러오기 기능

    for (let file_num = 0; file_num < profiles.length; file_num++) {
        profiles[file_num][0][0] = profiles[file_num][0][0].toLowerCase().slice(0,-4);
 //소문자로 core,task명 전부 바꾸고, 파일 확장자 제거

        // 이미 존재하는 Table이면 넘기는 처리를 하는 부분
        if (tableList.includes(profiles[file_num][0][0])) {
          console.log("이미 존재하는 파일입니다");
          continue;
        }

        await createDynamicTable(profiles[file_num]);        //동적 테이블을 생성(Core,Task의 갯수에 따라서)
        count++;
    }
        //count는 몇개의 input 파일을 처리했는지를 의미한다.(input 파일 여러개 처리 가능함)
        if(count===0){
            res.json({ status: 'success', message: `저장 가능한 파일이 존재하지 않습니다.` });
        }else if(count==profiles.length){
            res.json({ status: 'success', message: `${count}개의 프로파일이 정상적으로 저장되었습니다.` });
        }else{
            res.json({ status: 'success', message: `중복된 이름의 파일을 제외한 ${count}개의 프로파일이 저장되었습니다.` });
```

```
        }


        // 오류 발생시 처리
    } catch (error) {
        console.error('오류가 발생하였습니다:', error);
        res.json({ status: 'error', message: '오류가 발생하였습니다.' });
    }
});

// DB에서 table 목록 전체를 불러오고, Json 문서 형식으로 변환해서 응답하는 부분
router.get('/', async (req,res)=>{
    const tableList = await getTableList();
    res.json(tableList);
});

// 해당 테이블 명을 가진 Table을 호출하는 부분이다.(해당 inputfile을 클릭시, 불러오는 부분)
router.get('/data/:tableName', async (req,res)=>{
    try{
        const {tableName} = req.params;

        const tableList = await getTableList();     //1개의 테이블을 조회

        // 해당 table이 db에 존재하지 않으면, 오류 처리
        if(!tableList.includes(tableName)){
            return res.status(404).json({error:'존재하지 않는 파일입니다.'});
        }

       //테이블 이름의 데이터를 가져온다. DB와 연결한다, 모델 생성 초기화
        profile_model.initiate(sequelize, tableName);

        // 테이블의 모든 데이터를 가져와서, datas에 저장함
        const datas = await profile_model.findAll();

        // task 기준 core처리 현황을 불러옴(
        const tasks = await profile_model.findAll({
            attributes: [sequelize.fn('DISTINCT', sequelize.col('core')), 'core'],
        });
```

```
        // core 기준 task처리 현황을 불러옴
        const cores = await profile_model.findAll({
            attributes: [sequelize.fn('DISTINCT', sequelize.col('task')), 'task'],
        });

        // json 문서 형태로 응답(모든 데이터, core기준 task데이터, task기준 core데이터)
        res.json({datas: datas, cores : cores, tasks : tasks}); //json 형식으로 반환
    }catch(error){      // 오류 발생시
        console.error('데이터 조회 오류', error);
    }
});

// 해당 테이블을 삭제하는 기능
router.delete('/drop/:tableName', async(req,res)=>{
    try{
        const {tableName} = req.params;
        dropTable(tableName);      // 클릭시 테이블 삭제하는 기능
        res.json({state:'success'}); //json 형태로 반환, 상태 성공으로 업데이트
    }catch(error){
        res.json({state:'error'}); //json 형태로 반환, 상태 에러로 업데이트
    }
});

// CORE 기준으로 TASK그래프 표기시 사용하는 데이터 가공처리(평균, 최소, 최대)실행 후 반환
router.get('/coredata/:tableName/:core', async(req,res)=>{

    const { tableName, core } = req.params;      //테이블명, core정보 가져옴

    profile_model.initiate(sequelize, tableName);      //테이블 이름의 데이터를 가져온다. DB와 연결한다. 모델 생성 초기화

    const data = await profile_model.findAll({
        attributes: [
          'task',
          [sequelize.fn('max', sequelize.col('usaged')), 'max_usaged'], //최댓값처리
          [sequelize.fn('min', sequelize.col('usaged')), 'min_usaged'], //최솟값처리
          [sequelize.fn('avg', sequelize.col('usaged')), 'avg_usaged'] //평균값처리
        ],
```

```
        where: {
          core: core //core 값 선택
        },
        group: ['task'] //태스크별로 그룹화
      });

    res.json(data); //json으로 반환
});


// TASK 기준으로 CORE그래프 표기시 사용하는 데이터 가공처리(평균, 최소, 최대)실행 후 반환
router.get('/taskdata/:tableName/:task', async(req,res)=>{
    const { tableName, task } = req.params;

    profile_model.initiate(sequelize, tableName); //테이블 이름의 데이터를 가져온다. DB와 연결한다. 모델 생성 초기화


    const data = await profile_model.findAll({
        attributes: [
          'core',
          [sequelize.fn('max', sequelize.col('usaged')), 'max_usaged'], //최댓값처리
          [sequelize.fn('min', sequelize.col('usaged')), 'min_usaged'], //최솟값처리
          [sequelize.fn('avg', sequelize.col('usaged')), 'avg_usaged'] //평균값처리
        ],
        where: {
          task: task, //task 값만 선택
        },
        group: ['core'] //코어별로 그룹화
      });

    res.json(data); //json으로 반환
});


module.exports = router;     //이벤트 핸들링 처리지침이 담겨있는 router를 배포함
```

app.js

서버 설정 후 구동

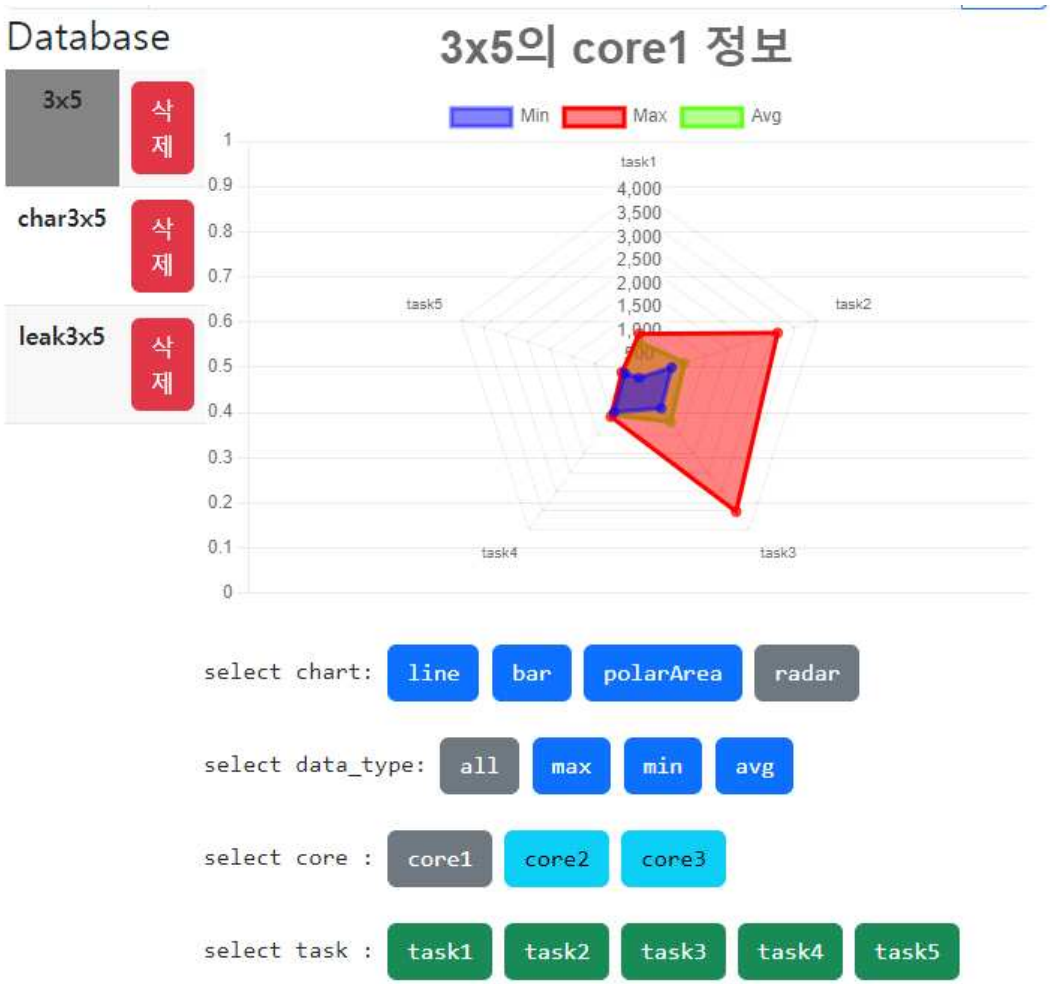
```
const express = require('express');     //서버 구성 편리 모듈
const morgan = require('morgan');       //파일 입출력 모듈
const path = require('path');     //경로설정 모듈
const nunjucks = require('nunjucks');     //HTML에서 템플릿 문법 사용(FOR문등 사용 가능)

const indexRouter = require('./routes');       //이벤트 감지-콜백 매핑 정보 저장
const profilesRouter = require('./routes/profiles');     //profile 모듈 가져오기
const {sequelize} = require('./models');     //models 모듈 가져오기



const app = express();     //서버 초기화
app.set('port', process.env.PORT || 3000);     //포트번호 3000번 설정
app.set('view engine', 'html');     // html 읽기가능 설정

// 넌적스 템플릿 문법 사용 가능하기 위한 조치
nunjucks.configure('views',{
    express: app,     //express랑 넌적스가 함께 동작하게 설정함
    watch: true,     //템플릿 코드 변경시 새로고침해서 변경사항이 반영
});

// 시퀄라이즈 동기화로 작동(이거 작동 되기 전까지는 아무도 못움직임), DB 불러오는 과정 인듯
sequelize.sync({force : false})     //{force : false}: 기존 존재 테이블 삭제 후 그 자리를 새 파일이 차지하게 설정 + DB연결 과정
    .then(()=>{
        console.log('데이터베이스 연결 성공');
    })
    .catch((err)=>{
        console.error(err);
    })

//app.use(morgan('dev'));     //개발자용 morgan 모듈 장착(HTTP 활동 로그 남김)
app.use(express.static(path.join(__dirname,'public')));       //정적 파일 전달을 위해 사용(CSS, JS파일 제공하는데 쓰임)
app.use(express.json());     //JSON형식 요청 처리를 express에서 하기 위해 사용함
app.use(express.urlencoded({extended: false}));     //URL 형식 요청 바디를 사용하기 위해 사용
```

```
app.use('/',indexRouter);          //메인페이지 진입시 기본적 라우팅('/' 경로 진입시, indexRouter 실행됨)
app.use('/profiles',profilesRouter);      //'/profiles' 경로 진입시, profilesRouter 실행됨

// 잘못된 URL 진입시 오류메시지 출력하는 부분
app.use((req,res,next)=>{
    const error = new Error(`${req.url}은 잘못된 주소입니다.`);
    error.status = 404;
    next(error);
});

//서버 문제 발생시 500번때 에러 status code 실행
app.use((err,req,res,next)=>{
    res.locals.message = err.message;
    res.status(err.status || 500);
    res.render('error');
});

//3000번 포트를 활용해 서버 가동하기
app.listen(app.get('port'), () =>{
    console.log("http://localhost:"+app.get('port')+" server open");
});
```

3. 기능 추가

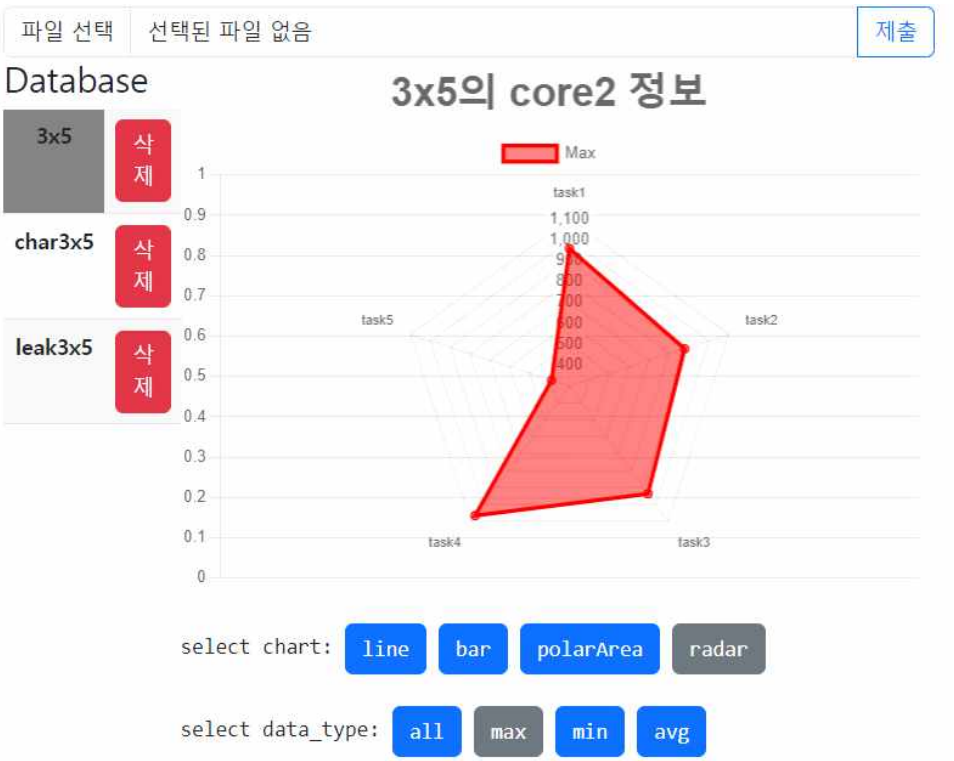
이 프로젝트에 핵심 기능은 Visualization이라 생각하여 Visualization을 높여줄 수 있는 아래와 같은 기능을 추가했다.

3.1 차트 형태 추가



새로운 차트 형태를 추가하여 다양한 방법으로 차트를 한 눈에 확인 할 수 있게 했다.

3.2 최댓값, 최솟값, 평균값 추가



최댓값, 최솟값, 평균값 기능을 넣어 원하는 정보만 볼 수 있도록 했다.

```
if(fileName==undefined || select==undefined) return;

let dataset = [];
if (data_type == 'all' || data_type == 'min') { //data type all or min이면
    dataset.push({ //min data 추가
        label: 'Min',
        data: minData,
        borderColor: 'rgba(0, 0, 255, 0.5)',
        backgroundColor: 'rgba(0, 0, 255, 0.5)',
    });
}
if (data_type == 'all' || data_type == 'max') { //data type all or max이면
    dataset.push({ //max data 추가
        label: 'Max',
        data: maxData,
        borderColor: 'rgba(255, 0, 0, 1)',
        backgroundColor: 'rgba(255, 0, 0, 0.5)',
    });
}
if (data_type == 'all' || data_type == 'avg') { //data type all or avg이면
    dataset.push({ //avg data 추가
        label: 'Avg',
        data: avgData,
        borderColor: 'rgba(100, 255, 30, 1)',
        backgroundColor: 'rgba(100, 255, 30, 0.5)',
    });
}
```

클릭한 data_type에 따라서 관련 데이터만 dataset에 넣어 차트를 출력한다.