



웹응용기술

# Core Task Profiler

과 목	웹응용기술
담당 교수	강영명 교수
팀 원	20210854 오현영

## <목차>

1. 프로그램 개요 .....	4
2. 프로그램 수행 절차 .....	4
2.1. 사용방법 안내 .....	4
2.2. 프로그램 시작 .....	5
2.3. 데이터 입력 .....	7
2.4. 차트 출력 및 기능 .....	7
2.5. 데이터 삭제 .....	8
3. 프로그램 기능 .....	9
3.1. Node 서버 구성 .....	9
3.2. Profile 파일 업로드 및 데이터 저장 .....	10
3.3. 저장된 프로파일 목록 조회 .....	11
3.4. 특정 프로파일 데이터 조회 및 시각화 .....	11
3.5. 차트 필터링 및 그룹화 .....	12
3.6. 통계 기능 .....	12
4. 후기 .....	13

## <그림 목차>

그림 1 config/config.json .....	4
그림 2 데이터베이스 생성 .....	5
그림 3 프로젝트 실행 .....	5
그림 4 웹 페이지 UI .....	6
그림 5 파일 업로드 .....	7
그림 6 업로드 성공 화면 .....	7
그림 7 차트 UI .....	7
그림 8 통계 분석 출력 .....	7
그림 9 데이터 삭제 기능 .....	8
그림 10 프로젝트 폴더 .....	9
[표 1] config, models, public, routes, views 폴더 .....	10

## 1. 프로그램 개요

CPU Core 는 프로세서 내에서 독립적으로 작업(Task)을 처리할 수 있는 물리적 단위이다. CPU Core 가 많을수록 많은 작업을 동시에 처리할 수 있다. Task 는 컴퓨터에서 수행되는 구체적인 작업이나 프로그램을 의미하므로 Core 가 해결할 수 있는 각 작업이 바로 이 task 가 된다.

본 프로젝트에서는 사용자가 텍스트 형식의 프로파일 데이터를 업로드하면 해당 데이터를 파싱, 가공된 데이터를 데이터베이스에 저장, 저장된 데이터를 호출 등의 과정을 거쳐 웹 UI 를 통해 사용자가 각 Core 에 대한 Task 를 다양한 시각화 차트로 확인 및 분석할 수 있는 "Core Task Profiler" 프로그램을 제안한다.

## 2. 프로그램 수행 절차

본 프로그램은 클라이언트(웹 브라우저)와 서버(Node.js 애플리케이션) 간의 상호작용으로 이루어진다. 사용자는 웹 UI 를 통해 파일을 업로드하거나 기존 데이터를 조회하며, 이 모든 요청은 HTTP 통신을 통해 서버로 전달된다. 서버는 요청을 처리하고 데이터베이스와 연동하여 필요한 작업을 수행한 후, 결과를 클라이언트에 응답한다

### 2.1 사용 방법 안내 (config.config.json 설정)

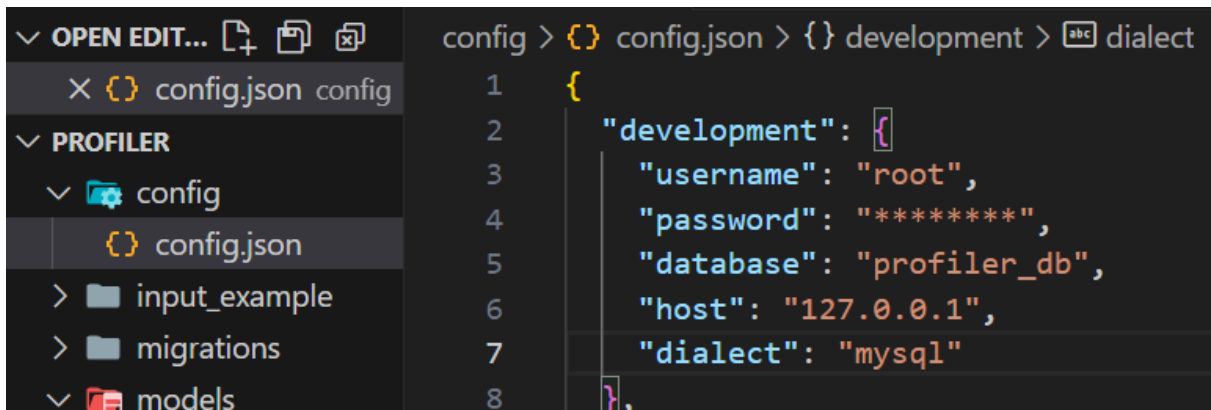


그림 1 config/config.json

해당 폴더의 config/config.json 파일로 이동하여 현재 입력되어 있는 password 를 사용자 컴퓨터에 설치된 MySQL 비밀번호로 변경한다.

```
mysql> mysql -u root -p
-> use profiler_db
```

그림 2 데이터베이스 생성

터미널에서 root 계정으로 MySQL에 접속한 후 "profiler\_db"를 생성하고 사용한다, "DataBase" 생성 완료 메시지가 콘솔에 출력되면 exit 명령어를 통해 MySQL을 종료한다.

```
$ npm start

> profiler@1.0.0 start
> nodemon

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
3001 번 포트에서 서버 대기 중
Executing (default): SELECT 1+1 AS result
DB 연결 성공
```

그림 3 프로젝트 실행

Npm install를 통해 node\_modules 폴더에 있는 모든 패키지를 다운로드 받은 후에, npm start 명령어를 통해 프로젝트를 실행시킨다.

## 2.2 프로그램 시작

현재 localhost에서 실행중이므로 localhost:3001로 접속하면 프로그램의 메인 페이지가 로드되고, 기존에 저장된 프로파일 목록이 있다면 자동으로 표시된다.

## Core Task Profiler

### 새 프로파일 업로드

프로파일 파일 선택 (.txt)

업로드

### 저장된 프로파일 목록

저장된 프로파일이 없습니다.

### 프로파일 차트

현재 프로파일: 선택되지 않음

그림 4 웹 페이지 UI

## 2.3 데이터 입력

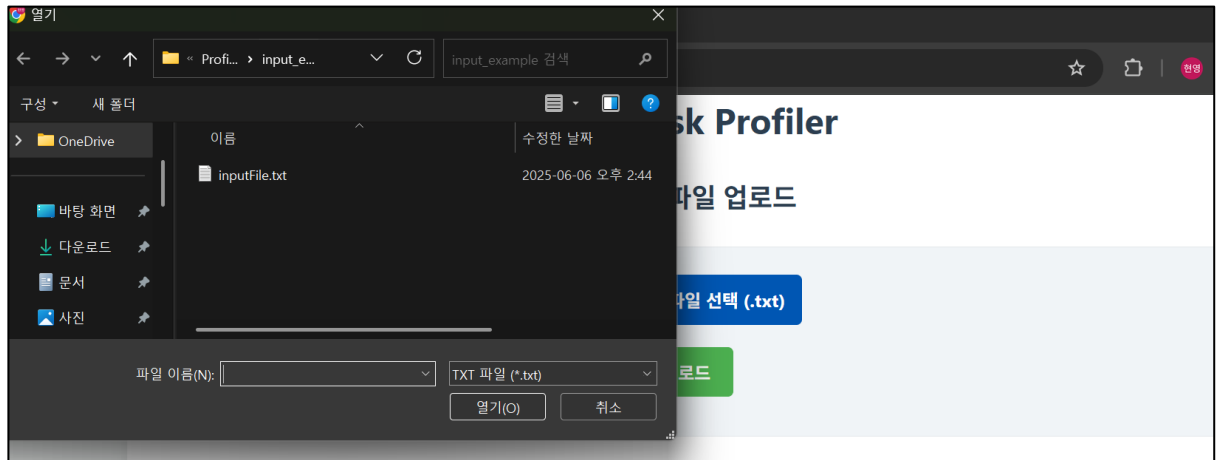


그림 5 파일 업로드

“새 프로파일 업로드” 섹션에서 “프로파일 파일 선택 (.txt)” 버튼을 클릭하여 로컬 컴퓨터에 저장된 하나 이상의 '.txt' 프로파일 파일을 선택한 후 업로드한다.

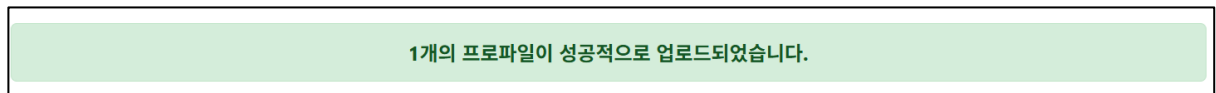
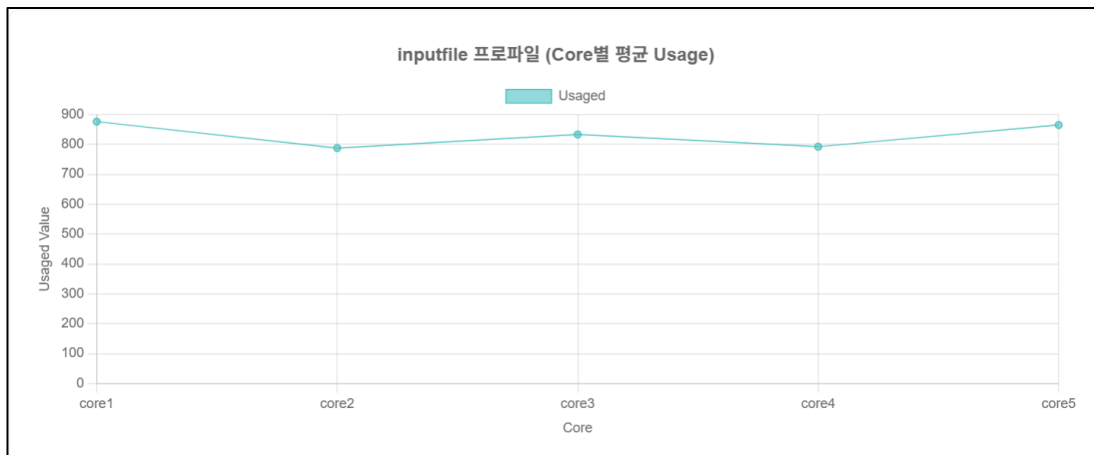


그림 6 업로드 성공 화면

## 2.4 결과 출력 및 기능



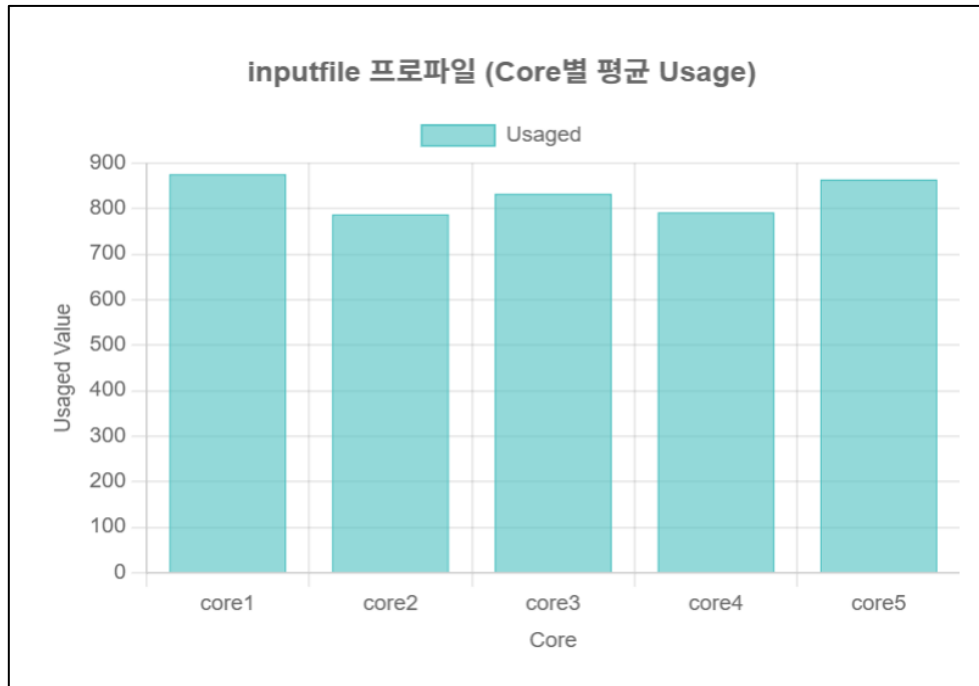


그림 7 차트 UI



그림 8 통계 분석 출력



“저장된 프로파일 목록”에서 특정 프로파일의 “조회” 버튼을 클릭하면 해당 프로파일의 데이터가 하단의 “프로파일 차트” 섹션에 로드된다. 드롭다운을 통해 ‘막대 차트 (Bar)’, ‘선 차트 (Line)’, ‘극좌표 차트 (Polar Area)’ 중 원하는 시각화 형태를 선택할 수 있다. “집계 기준” 역시 드롭다운 기능을 통해 Core 와 Task 별로 선택 가능하며 개별적으로 필터링할 수 있다. 옵션을 변경한 후 “차트 업데이트” 버튼을 클릭하면 새로운 옵션에 맞춰 동적으로 다시 그려진다.

## 2.5 데이터 삭제

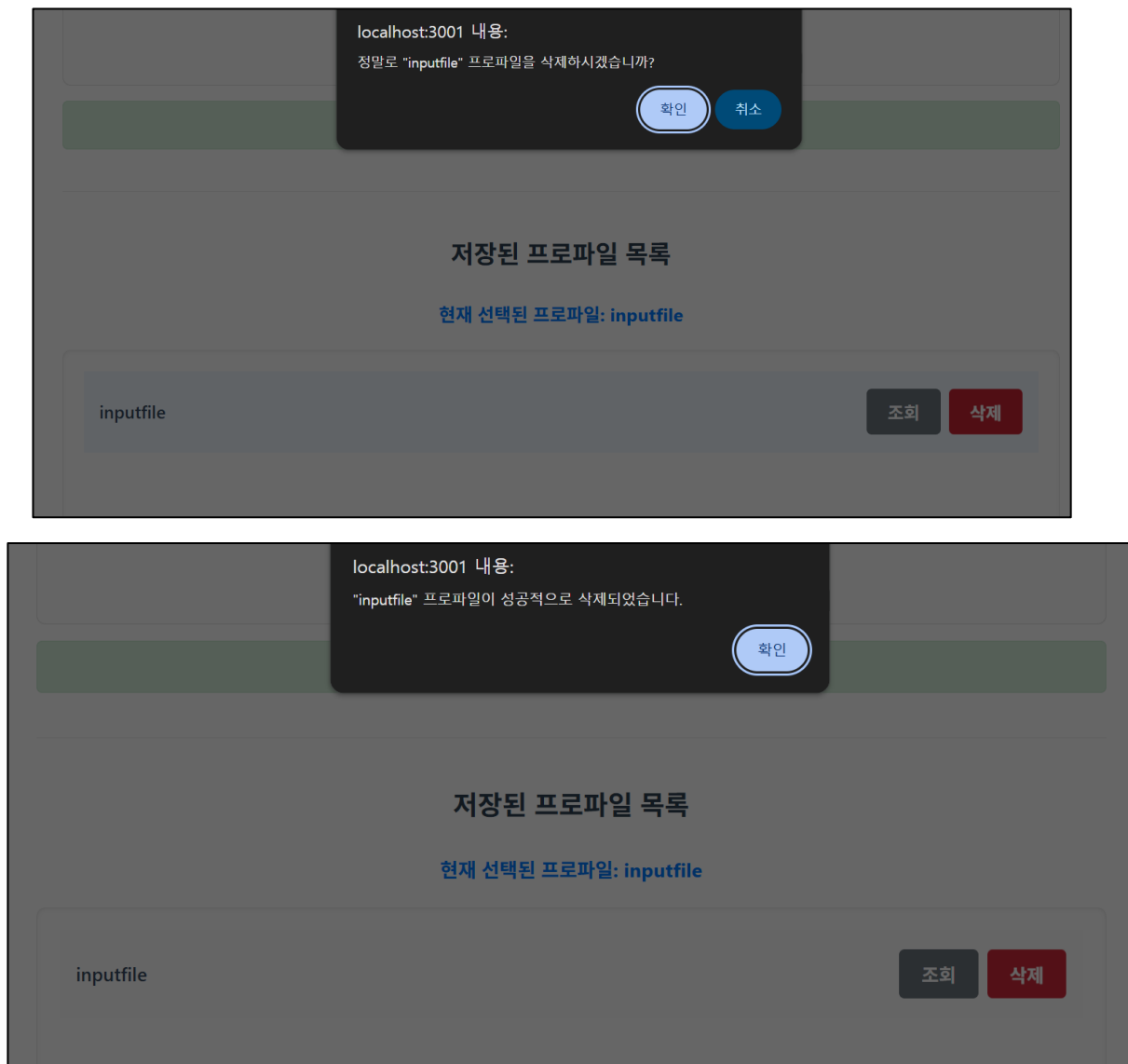


그림 8 데이터 삭제 기능

"저장된 프로파일 목록"에서 삭제하고자 하는 프로파일의 오른쪽에 위치한 "삭제" 버튼을 클릭한다. "확인" 버튼을 클릭하면 서버로 삭제 요청이 전송되고, 해당 프로파일 데이터베이스 테이블이 영구적으로 삭제됩니다.

### 3. 프로그램 기능

#### 3.1 Node 서버 구성(폴더 구성 설명)

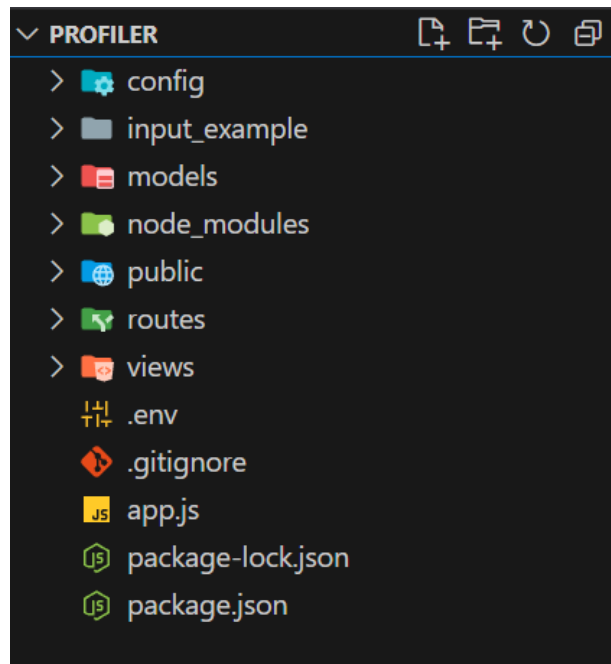


그림 9 프로젝트 폴더

Express.js 프레임워크를 활용하여 안정적인 웹 서버 환경을 구축한다. app.js에서는 서버 포트, 뷰 엔진(Nunjucks), 정적 파일 서비스 처리와 같은 여러가지 미들웨어를 설정한다. Sequelize ORM을 통해 MySQL 데이터베이스와 연결하고, 서버 시작 시 데이터베이스 스키마를 동기화하여 데이터베이스와 애플리케이션 간의 연동을 초기화한다.

[표 1] config, models, public, routes, views 폴더

폴더명	주요 역할	상세 설명
Routes/	라우터 정의	클라이언트의 HTTP 요청을 처리하는 경로별 라우터 파일을 포함한다.
Models/	데이터베이스 모델 정의	데이터베이스 연결 설정, Sequelize ORM 모델 정의, 그리고 동적 테이블 생성/관리 및 데이터베이스 상호작용 로직을 포함하는 폴더.
Public/	정적 파일	웹 브라우저에서 직접 접근하여 사용하는 클라이언트 측 정적 파일을 처리하는 폴더
views/	템플릿 파일	서버에서 동적으로 HTML 페이지를 렌더링하기 위한 Nunjucks 템플릿 파일들을 포함하는 폴더
config/	환경 설정	데이터베이스 연결 정보와 같은 환경 설정 값을 저장 파일

### 3.2. Profile 파일 업로드 및 데이터 저장

사용자가 웹 UI의 "새 프로파일 업로드" 섹션에서 하나 이상의 .txt 파일을 선택하고 "업로드" 버튼을 클릭한다. 클라이언트 측 JavaScript(public/main.js)는 선택된 파일들의 내용을 비동기적으로 읽어들이고, 각 파일의 첫 줄을 테이블 이름으로 사용하고 나머지 줄들의 core, task, usage 값을 파싱한다. 이 파싱된 데이터는 [[테이블\_이름], [core, task, usage], ...] 형태의 구조화된 JSON 배열로 가공되어 HTTP POST 요청의 본문(req.body)을 통해 서버의 /profiles 엔드포인트로 전송된다.

서버(routes/profiles.js)는 이 데이터를 수신하여 models/index.js의 createDynamicTable 함수를 호출한다.

createDynamicTable 함수는 파일 이름에서 추출된 테이블 이름을 기반으로 데이터베이스에 새로운 테이블을 동적으로 생성하고, 파싱된 모든 프로파일 데이터를 Sequelize.Model.bulkCreate() 메서드를 이용하여 해당 테이블에 효율적으로 저장하고 이 과정에서 이미 존재하는 테이블 이름은 건너뛰어 데이터 중복을 방지한다.

### 3.3. 저장된 프로파일 목록 조회

데이터베이스에 현재 저장되어 있는 모든 동적 프로파일 테이블의 이름을 조회하여 클라이언트에 제공한다. 사용자가 애플리케이션에 처음 접속하거나 페이지를 새로 고칠 때, routes/index.js는 models/index.js의 getTableList() 함수를 호출하여 데이터베이스의

information\_schema를 쿼리하여 테이블 목록을 가져온다. 이 목록은 Nunjucks 템플릿 (views/index.html)으로 전달되어 "저장된 프로파일 목록" UI를 초기 렌더링한다. 또한, 파일 업로드 성공 후에는 클라이언트(public/main.js)가 routes/profiles.js의 GET /profiles API를 호출하여 최신 테이블 목록을 비동기적으로 요청하고, 응답받은 JSON 데이터를 바탕으로 UI를 동적으로 갱신한다. 이를 통해 사용자는 현재 관리 가능한 프로파일들을 항상 최신 상태로 파악할 수 있다.

### 3.4. 특정 프로파일 데이터 조회 및 시각화

"저장된 프로파일 목록"에서 사용자가 특정 프로파일의 "조회" 버튼을 클릭하면, 클라이언트(public/main.js)는 해당 프로파일의 테이블 이름을 URL 파라미터로 포함하여 routes/profiles.js의 GET /profiles/:tableName를 호출한다. 서버는 req.params에서 테이블 이름을 추출한 후, models/index.js의 getDynamicModel(tableName) 함수를 사용하여 해당 테이블에 대한 Sequelize 모델 인스턴스를 가져온다. 이 모델을 통해 core, task, usaged 컬럼의 모든 데이터를 조회하여 클라이언트에 JSON 형태로 반환한다. 클라이언트는 응답받은 데이터를 전역 변수에 저장하고, Chart.js 라이브러리를 활용하여 막대, 선, 극좌표 등 다양한 차트 형태로 데이터를 시각적으로 표현하여 사용자에게 직관적인 분석 결과를 제공한다.

### 3.5. 차트 필터링 및 그룹화

시각화된 데이터에 대해 유연한 분석 기능을 제공하여 사용자가 원하는 방식으로 데이터를 탐색할 수 있도록 돕는다. 사용자는 "차트 종류" 드롭다운을 통해 '막대', '선', '극좌표' 차트 중 하나를 선택하여 데이터 표현 방식을 변경할 수 있다. 더불어, "집계 기준" 드롭다운을 통해 데이터를 'Core별' 또는 'Task별'로 그룹화하여 해당 그룹의 usaged 총합을 확인할 수 있다. 'Core별' 또는 'Task별' 집계 기준을 선택할 경우, 해당 Core 또는 Task 값들을 담은 개별 필터링 드롭다운(Core 선택, Task 선택)이 동적으로 활성화되어 특정 Core 또는 Task의 데이터만을 선택적으로 분석할 수 있도록 한다. 모든 필터링 및 그룹화 조건 변경 후 "차트 업데이트" 버튼을 클릭하면 public/main.js의 drawChart 함수가 재실행되어 새로운 조건에 맞춰 차트가 동적으로 다시 그려진다.

### 3.6. 통계 기능

통계 기능은 사용자가 선택한 프로파일 데이터를 기반으로 전체, Core별, Task별 used 값의 최소, 최대, 평균, 표준편차를 동적으로 계산한다. 클라이언트 (public/main.js)는 선택된 프로파일과 통계 유형에 따라 서버(routes/profiles.js)에 API 요청을 보내면 서버는 요청받은 테이블에 대해 models/index.js의 동적 모델을 사용하여 Sequelize의 집계 함수(fn, literal, group)를 통해 SQL 쿼리를 실행하고, 결과 값을 숫자로 변환하여 클라이언트에 응답한다. 클라이언트는 응답 받은 통계 데이터를 안전하게 파싱하여 웹페이지에 시각적으로 표시한다.

## 4. 후기

기존에 개발된 프로파일러 프로그램을 기반으로 확장 기능을 구현하는 과정에서 프론트 엔드와 백엔드 모두에서 다양한 기술적 난관에 직면했다. 특히 통계 기능 구현 중 avg 값이 문자열로 반환되어 웹 페이지에 로딩되지 않거나, 데이터베이스가 존재함에도 불구하고 서버가 이를 인식하지 못하는 등 여러 문제들을 해결해야 했다. 비록 프로그램의 완성도가 아직 높지 않지만, 이러한 문제들을 극복하고 최종 결과물을 도출해냈다는 점에서 큰 성취감을 느꼈고 동시에 아직 부족한 점이 많다는 것을 깨달았다. 앞으로는 더욱 심층적인 프로그램 개발을 목표로 심도 있는 학습을 이어나갈 계획이다.