Node.js를 활용한 Profiler 프로그램 분석

학과	컴퓨터공학과
학번	20210840
이름	박예은
Github 링크	https://github.com/yeni023/profiler-project
Notion 링크	https://piquant-beryllium-6b0.notion.site/Node-js-Profiler-20d872c7c8b68023946ed22fc94e245f?source=copy_link

1. 프로그램 개요

오늘날의 컴퓨팅 환경에서는 하나의 작업이 아닌 여러 작업을 동시에 처리하는 병렬 처리 성능이 핵심 경쟁력으로 떠오르고 있다. 특히 여러 개의 CPU 코어에서 수행되는 작업(Task)의 성능을 정량적으로 분석하고 시각화하는 일은 시스템 최적화와 병목 지점 파악에 필수적이다.

본 프로젝트는 Node.js와 웹 브라우저 기반 기술을 활용하여, 사용자가 업로드한 데이터 파일을 바탕으로 CPU 코어(Core) 및 작업(Task) 단위의 성능 정보를 분석하고 시각화하는 웹 기반 성능 프로파일러를 구현하였다.
사용자는 웹 페이지를 통해 데이터를 업로드하며, 서버는 이를 분석해 최소(MIN), 최대(MAX), 평균(AVG) 등의 기초 통계를 산출하고, 선택적으로 표준편차(Standard Deviation) 분석도 지원한다. 이렇게 처리된 데이터는 데이터베이스에 저장되며, 이후 브라우저 상에서 차트 형태로 시각화된다.
이 프로젝트는 단순한 데이터 분석에 그치지 않고, 업로드된 파일 목록 관리, 파일별 분석 결과 비교 등 실용적인 기능을 포함한다. 이를 통해 사용자는 CPU 작업 데이터를 직관적으로 이해하고, 보다 효율적인 성능 분석 환경을 경험할 수 있다.

1.1 주요 기능 요약

기능	설명
파일 업로드	사용자가 .txt 형식의 파일을 업로드하여 서버에 저장
동적 테이블 생성	업로드한 파일명을 기준으로 데이터베이스에 별도 테이블 생성
통계 분석	AVG, MIN, MAX, STDDEV 등의 통계값 산출
차트 시각화	Chart.js를 이용한 Bar, Line, PolarArea 등의 그래프 지원
버튼 자동 생성	core, task, 통계 기준, 차트 유형 선택 버튼을 동적으로 생성
다중 파일 처리	여러 파일을 동시에 업로드 및 비교 가능
파일별 삭제 기능	특정 파일의 데이터와 관련된 테이블을 개별 삭제 가능
사용자 알림	업로드/삭제 등의 작업 완료 후 toast 메시지로 안내

2. 전체 실행 흐름

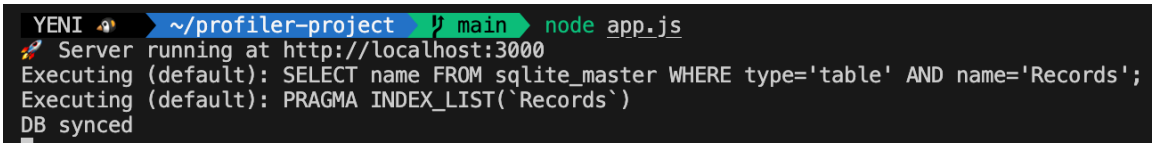
2.1 프로그램 시작 및 실행

본 프로젝트는 **Node.js 기반 백엔드 서버**와 **정적 HTML/JavaScript 프론트엔드**로 구성되어 있으며, 사용자는 웹 브라우저를 통해 애플리케이션에 접근할 수 있다.

프로젝트 실행은 다음과 같은 순서로 이루어진다.

1. 서버 실행
   터미널에서 프로젝트 디렉터리로 이동한 후,
   `node app.js` 명령어를 입력하여 서버를 실행한다.

서버는 기본적으로 http://localhost:3000에서 구동되며, 실행이 완료되면 콘솔에 `Server running at` `http://localhost:3000` 메시지가 출력된다.



서버 실행 시 출력되는 콘솔 화면

2. 클라이언트 접근
   웹 브라우저에서 http://localhost:3000에 접속하면, 초기 화면에 파일 업로드 인터페이스가 표시된다. 이후 사용자가 데이터를 업로드하면 서버가 이를 처리하고, 통계 결과를 시각화하여 출력한다.

2.2 데이터 업로드 처리

웹 페이지에 접속하면 화면 상단에 **파일 업로드 인터페이스**가 표시된다. 사용자는 `inputFile.txt`와 같은 텍스트 파일을 선택하여 업로드할 수 있으며, 이는 HTML의 `<form>` 요소와 `<input type="file">` 태그를 통해 구현되어 있다.

사용자가 파일을 선택한 뒤 업로드 버튼을 클릭하면, 클라이언트에서 서버로 **POST 요청**이 전송된다. 이 요청은 `/upload-dynamic` 경로로 전달되며, 백엔드에서는 **Multer 미들웨어**를 이용해 업로드된 파일을 서버 로컬 디렉터리에 저장한다.

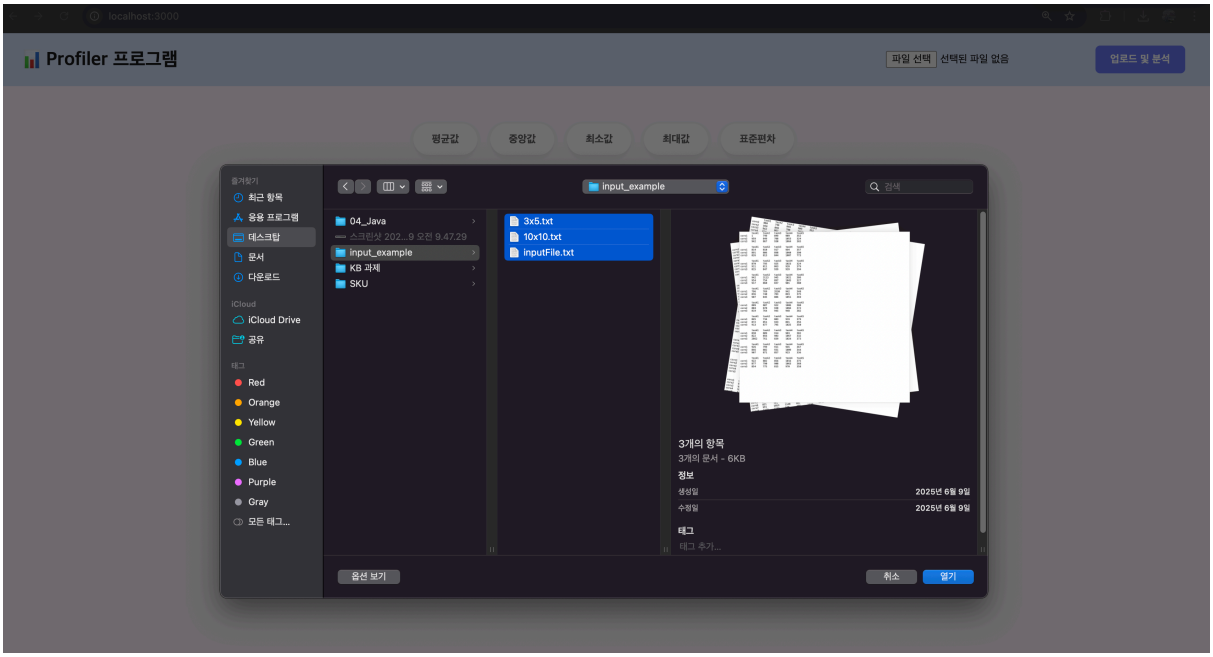
업로드된 파일은 서버에서 **한 줄씩 읽어 들인 후**, `core`, `task`, `usaged` 등의 필드로 데이터를 파싱한다. 이후 파싱된 데이터는 Sequelize를 통해 **파일명을 기반으로 동적으로 생성된 데이터베이스 테이블**에 저장된다. 이 방식을 통해 사용자는 여러 개의 파일을 업로드하더라도 각각의 데이터를 독립적으로 관리할 수 있다.

업로드가 완료되면, 클라이언트 화면에는 해당 파일 이름을 가진 **버튼이 자동으로 생성**된다. 사용자는 이 버튼을 클릭하여 해당 데이터의 분석 결과를 **차트 형태로 시각화**하여 확인할 수 있다.

해당 이미지는 사용자가 웹 브라우저로 애플리케이션에 접속했을 때 표시되는 초기 화면이다. 상단에는 텍스트 파일 업로드를 위한 입력 폼과 업로드 버튼이 위치하며, 그 아래로는 분석 기능을 선택할 수 있는 다양한 버튼과 차트 영역이 배치되어 있다.

웹 페이지 접속 화면

사용자는 하나 이상의 `.txt` 파일을 다중 선택할 수 있으며, 선택된 모든 파일은 서버에 동시에 업로드된다. 이 기능을 통해 여러 코어 작업 데이터를 한 번에 분석하고, 각각의 파일별로 독립적인 결과를 확인할 수 있도록 구현하였다.



파일 선택 (다중 선택 화면)

2.3 결과 시각화 출력

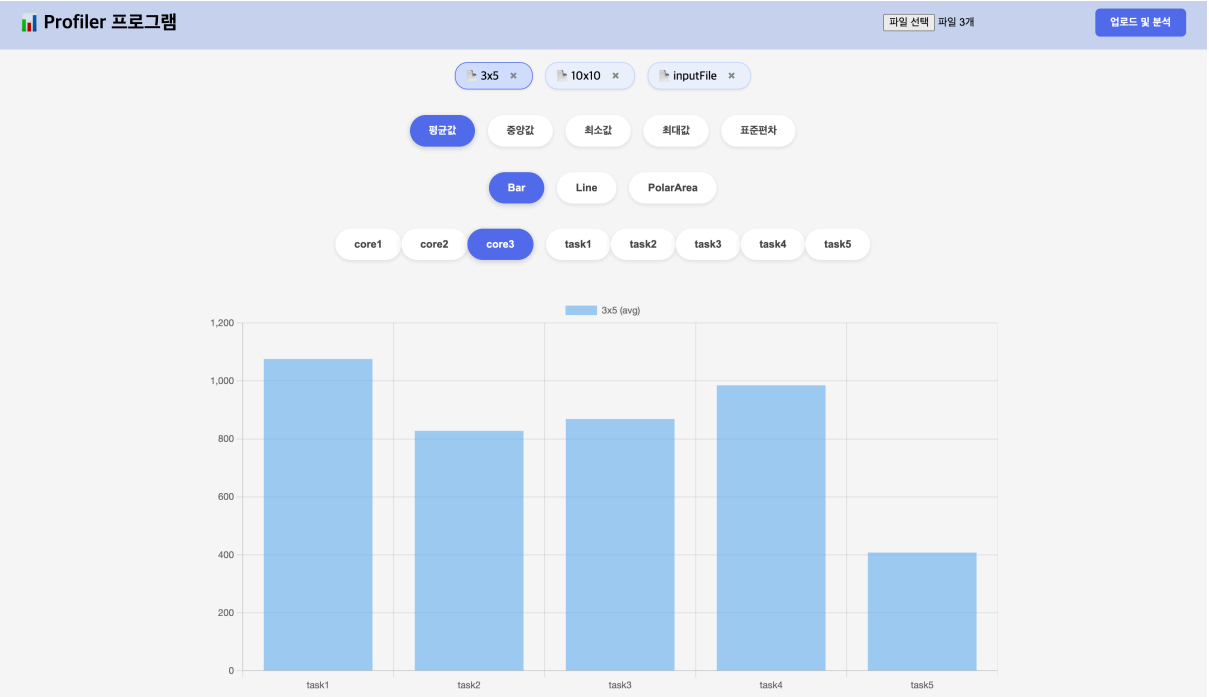
사용자가 파일 업로드를 완료하면, 클라이언트 화면에는 업로드된 파일 이름이 포함된 **버튼 목록**이 동적으로 생성된다. 각 버튼을 클릭하면 해당 파일의 분석 결과가 서버에서 조회되어, 화면 하단의 **차트 영역**에 시각적으로 출력된다.

분석 결과는 평균값(AVG), 중앙값(MEDIAN), 최소값(MIN), 최대값(MAX), 표준편차(STDDEV) 등의 통계 항목 중 선택된 기준에 따라 필터링되어 표시된다. 사용자는 원하는 항목을 선택하여 분석의 초점을 전환할 수 있다.

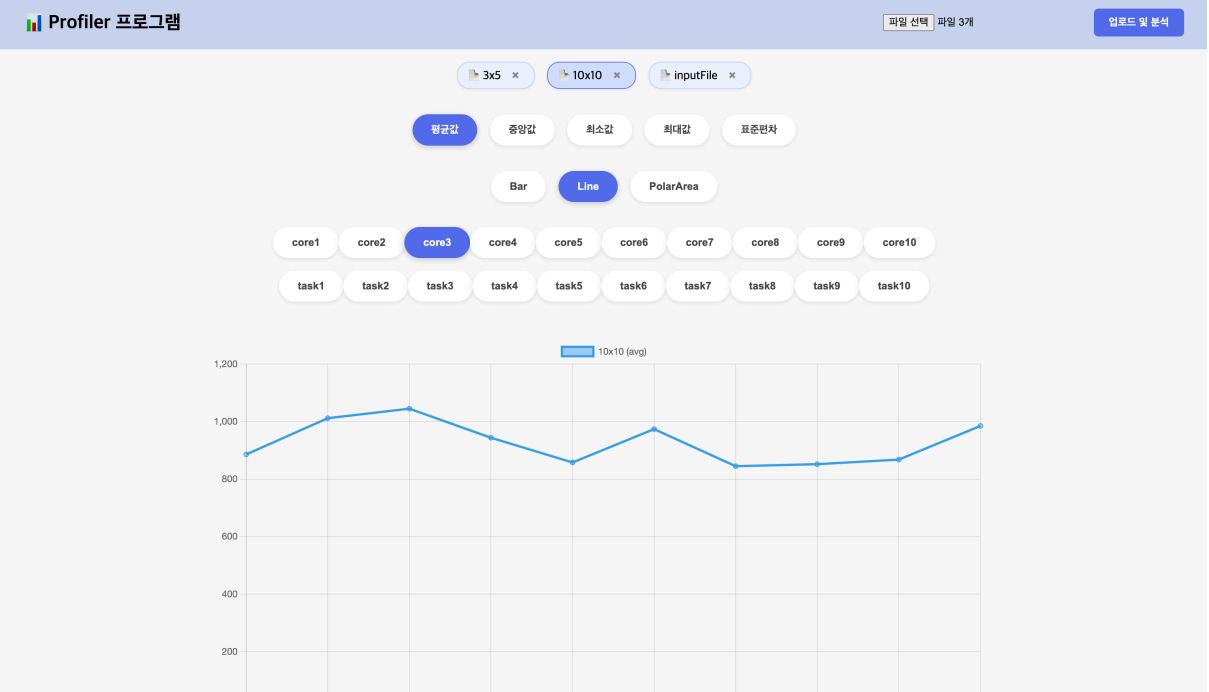
또한, 화면 상단에는 **그래프 형태를 선택할 수 있는 버튼 그룹**이 제공되어, 사용자는 막대 그래프(Bar), 선형 그래프(Line), 폴라 에어리어 차트(PolarArea) 중 원하는 방식으로 결과를 시각화할 수 있다.

이러한 시각화는 **Chart.js** 라이브러리를 활용해 동적으로 렌더링되며, 선택된 통계 기준과 그래프 유형에 따라 실시간으로 차트가 갱신된다.

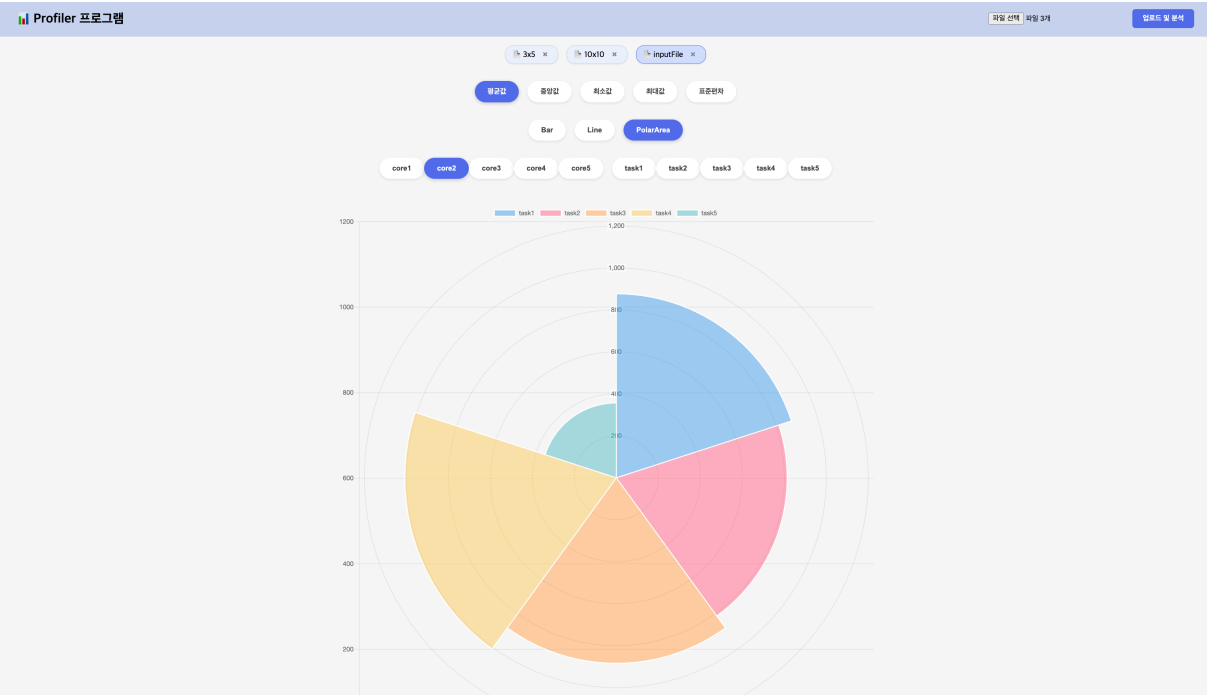
차트는 특정 코어(Core) 또는 태스크(Task)를 기준으로 그룹화하여 출력할 수 있으며, 각각의 선택지는 버튼 형태로 제공되어 사용자가 간편하게 전환할 수 있다. 선택한 파일, 기준(Core 또는 Task), 통계 항목, 그래프 유형을 조합함으로써, **사용자는 데이터를 직관적으로 분석하고 비교할 수 있는 시각화 환경**을 제공받게 된다.



3×5 데이터 프로파일링 결과 (Bar 형태)



10×10 데이터 프로파일링 결과 (Line 형태)



inputFile 데이터 프로파일링 결과 (PolarArea 형태)

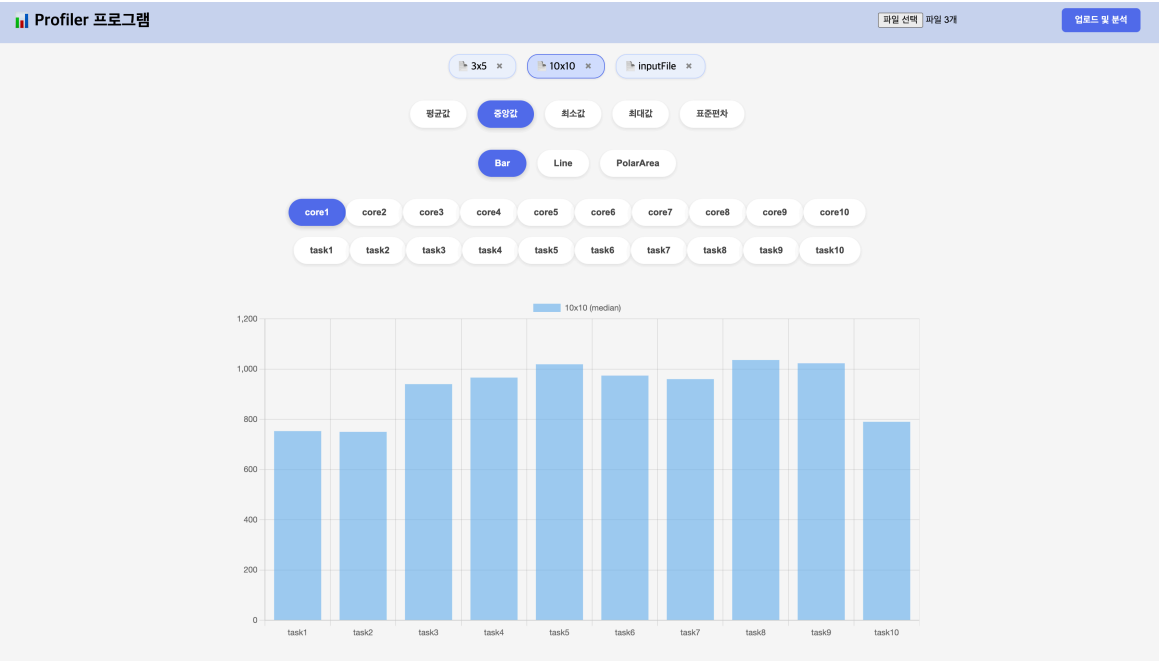
2.4 버튼 동적 생성

프로그램에서는 사용자의 입력과 서버 응답에 따라 다양한 버튼들이 동적으로 생성된다. 이를 통해 사용자 인터페이스가 직관적으로 유지되며, 분석할 대상을 유연하게 선택할 수 있도록 구성되어 있다.

업로드된 데이터에 포함된 `core`와 `task` 정보에 따라 클라이언트 화면에 다양한 버튼이 자동 생성된다. 이 버튼들을 통해 사용자는 **분석 대상 데이터**, **통계 기준**, **시각화 방식**을 자유롭게 선택할 수 있으며, 차트를 갱신하면서 결과를 직관적으로 확인할 수 있다.

예를 들어, `10x10` 형태의 데이터가 포함된 텍스트 파일을 업로드한 경우, 화면에는 다음과 같은 버튼들이 생성된다:

- **Core 선택 버튼:** core1 ~ core10 (총 10개)
- **Task 선택 버튼:** task1 ~ task10 (총 10개)
- **통계 기준 버튼:** 평균값(AVG), 중앙값(MEDIAN), 최소값(MIN), 최대값(MAX), 표준편차(STDDEV)
- **차트 유형 버튼:** Bar, Line, PolarArea



10×10 데이터 프로파일링 결과 (Bar 형태)

2.5 데이터 삭제 기능

프로그램에서는 사용자가 업로드한 각 파일을 개별적으로 관리할 수 있도록, **파일별 삭제 기능**을 제공한다. 업로드된 파일 이름 버튼 우측에는 `×` 표시의 삭제 버튼이 함께 표시되며, 이를 통해 사용자는 특정 파일에 대한 분석 데이터를 삭제할 수 있다.

사용자가 삭제 버튼을 클릭하면, 다음과 같은 절차로 삭제가 진행된다.

1. **클라이언트 요청**

   클라이언트에서는 해당 파일 이름을 기반으로 `/upload-dynamic/tables/:fileName` 경로로 **HTTP DELETE 요청**을 전송한다.

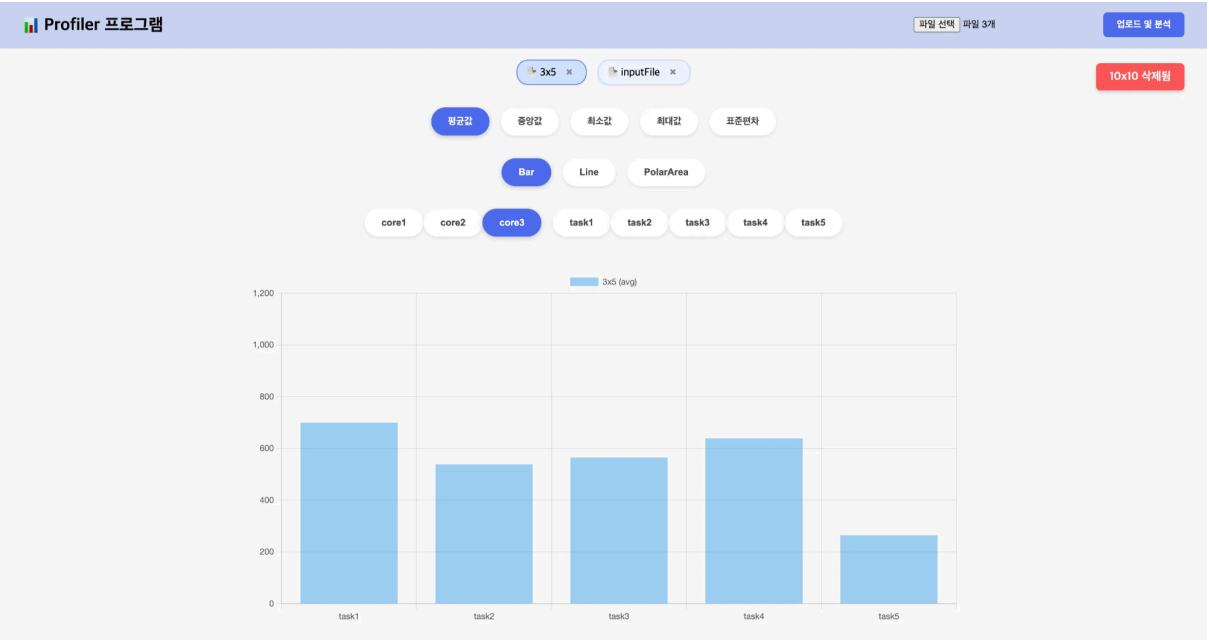
2. **서버 처리**

   서버에서는 전달받은 파일명을 기준으로 생성된 데이터베이스 테이블을 식별한 뒤, 해당 테이블을 제거한다. 이 작업은 `dropTable()` 함수에 의해 수행되며, 내부적으로는 SQLite 명령어 `DROP TABLE`이 실행된다.

3. **화면 반영**

   삭제가 완료되면, 클라이언트는 화면 상에서 해당 파일의 버튼을 제거하고, 선택된 파일이 삭제된 경우에는 다른 파일로 자동 전환하거나 차트를 초기화한다. 또한 삭제 완료 메시지는 **toast 알림 형태**로 사용자에게 제공된다.

이러한 삭제 기능을 통해 사용자는 **분석에 필요 없는 파일을 언제든지 제거할 수 있으며**, 여러 파일을 반복적으로 업로드하고 관리하는 데 있어 편의성을 확보할 수 있다.



10×10 데이터 삭제 화면

3. 프로그램 구조 분석

3.1 Node.js 서버 구성

이 프로젝트의 백엔드는 **Node.js**와 **Express.js** 기반으로 구성되어 있으며, 주요 기능은 다음과 같은 파일과 라우터를 중심으로 처리된다.

- `app.js` : 애플리케이션의 **진입점(entry point)** 역할을 하며, 전체 라우팅과 미들웨어 설정, 정적 파일 경로 및 서버 실행을 담당한다.

- `/routes/*` : 클라이언트 요청을 처리하는 **라우터 모듈**로, 기능별로 분리되어 있다.
  - `upload.js` : 단일 파일 업로드 및 기본 통계 조회를 처리
  - `upload-dynamic.js` : 다중 파일 업로드 및 동적 테이블 생성/삭제/조회 기능 제공
  - `dynamic.js` : 셀렉터 정보를 별도로 가져오는 전용 라우터
- `/controllers/*` : 실제 로직이 구현되어 있는 **컨트롤러** 모듈로, 라우터에서 분기된 요청을 처리
  - 파일 파싱, DB 저장, 통계 계산, 차트용 데이터 가공 등이 포함된다.
- `/models/*` : Sequelize를 사용한 **ORM 모델 정의 및 데이터베이스 연동** 모듈
  - `Record.js` : 정적 테이블용 모델
  - `dynamic.js` : 파일 기반 동적 테이블 생성 로직 포함
  - `index.js` : Sequelize 초기화 설정

> 서버는 포트 `3000` 에서 실행되며, 브라우저에서 `http://localhost:3000` 으로 접근할 수 있다.
> 정적 자산은
> `/public` 디렉터리를 통해 제공되며, HTML, CSS, JS, Chart.js 등의 자원들이 여기에 포함된다.

3.2 클라이언트 구조

프로젝트의 프론트엔드는 순수 **HTML, CSS, JavaScript(ES6 모듈)**로 구성되어 있으며, 주요 역할은 다음과 같이 분리되어 있다.

- `views/index.html`
  - 웹 애플리케이션의 기본 구조 제공
  - 업로드 폼, 파일 리스트, 버튼 그룹, 차트 영역 등을 포함
  - `Chart.js` 라이브러리와 여러 JS 모듈을 로드함
- `/public/js/upload.js`
  - 업로드 폼 처리 및 파일 전송
  - 업로드된 파일 목록 관리 및 삭제
  - 버튼 클릭 이벤트 바인딩 (차트 기준 및 형태 선택)

- `/public/js/selectors.js`
  - 서버에서 core/task 선택지를 불러와 버튼 렌더링
  - 사용자가 선택한 core/task 기준에 따라 차트 갱신 호출
- `/public/js/chart.js`
  - `Chart.js`를 사용해 선택된 데이터를 시각화
  - 현재 선택된 통계 기준 및 차트 타입에 따라 실시간으로 차트 구성
- `/public/js/toast.js`
  - 삭제 완료, 오류 등의 메시지를 사용자에게 toast 형태로 표시

> 각 모듈은 역할에 따라 책임 기반으로 분리되어 있으며, 유지보수성과 확장성을 고려한 구조로 작성되어 있다.

3.3 데이터베이스 구조

이 프로젝트에서는 **SQLite**를 데이터베이스로 사용하며, **Sequelize ORM**을 통해 데이터 저장, 조회, 삭제 등의 작업을 수행한다.

데이터베이스는 크게 두 가지 방식으로 데이터를 저장한다.

3.3.1 정적 테이블 (Record)

- 사용 경로: `/upload`를 통한 단일 파일 업로드 시
- 모델 정의: `models/Record.js`
- 저장 항목:
  - `core` (문자열)
  - `task` (문자열)
  - `value` (정수형)

이 테이블은 고정된 구조로, 모든 데이터가 동일한 테이블에 누적 저장된다. 이후 통계 계산(AVG, MIN, MAX 등)에 활용된다.

3.3.2 동적 테이블 (파일 기반 생성)

- 사용 경로: `/upload-dynamic`을 통한 다중 파일 업로드 시

- 생성 방식: 업로드된 **파일명을 기반으로 동적 테이블 생성**
- 모델 로직: `models/dynamic.js` 내 `createDynamicTable()` 함수
- 저장 항목:
  - `core` (문자열)
  - `task` (문자열)
  - `usaged` (정수형)

⬇️ 예시 테이블 구조 (`inputFile` 업로드 시)

id (PK)	core	task	usaged
1	core1	task1	82
2	core2	task2	76
3	core3	task3	91

> 테이블 이름은 업로드한 파일명을 기준으로 자동 생성되며, 각 파일의 데이터는 독립적으로 저장된다.

이 방식은 업로드된 각 파일의 데이터를 별도의 테이블에 저장함으로써, **파일별로 데이터를 독립적으로 관리**

할 수 있게 해준다.

사용자는 특정 파일에 대한 분석 결과를 손쉽게 조회하거나 삭제할 수 있어, 다중 파일 기반 분석 환경에서 높은 유연성과 편의성을 확보할 수 있다.

3.3.3 관리용 쿼리 및 함수

데이터베이스에 저장된 정보를 조회하거나 정리하기 위해 아래와 같은 유틸리티 함수들이 구현되어 있다.

- **조회용**
  - `getTableList()` : 현재 존재하는 동적 테이블 목록 반환
  - `getChartData()` : core/task 기준의 집계 통계값 조회
  - `getSelectors()` : 테이블에 존재하는 core/task 항목 추출
- **삭제용**

- `dropTable(tableName)` : 동적으로 생성된 테이블 제거

3.4 프론트엔드 구성 요소

이 프로젝트의 프론트엔드는 순수 HTML, CSS, JavaScript로 구성되어 있으며, **정적 자산은 `/public` 디렉터리에 위치**한다. 사용자는 웹 브라우저를 통해 다양한 인터랙션을 직관적으로 수행할 수 있도록 구성되어 있다.

주요 파일 및 역할

파일명	설명
`index.html`	메인 화면을 구성하는 HTML 파일, 파일 업로드 인터페이스 버튼 영역, 차트 영역 등을 포함
`upload.js`	파일 업로드와 관련된 로직 처리 (form 데이터 전송 등)
`selectors.js`	core/task 선택 버튼 로직 처리 및 선택 값 전달
`chart.js`	Chart.js를 활용해 시각화 차트를 렌더링하는 모듈
`toast.js`	파일 삭제/업로드 완료시 사용자 알림 메시지 출력
`style.css`	기본 UI 스타일 정의

프론트엔드 흐름 요약

1. 사용자가 파일을 업로드하면 `upload.js`가 해당 파일을 서버로 전송
2. 서버 응답 후 파일 목록 버튼이 생성되고, 사용자는 이를 클릭해 차트를 요청
3. 사용자가 core/task/stat/graph 유형을 선택하면, `selectors.js`가 선택값을 전달
4. `chart.js`는 선택된 옵션에 따라 실시간으로 차트를 렌더링
5. 삭제 시에는 `toast.js`를 통해 알림 메시지를 표시하며 UI를 갱신

> 이와 같은 구조는 **JS 파일이 기능별로 모듈화**되어 있어 유지보수와 기능 확장이 용이하며, 구조적 이해에도 도움이 된다.

3.5 차트 출력 과정

본 프로젝트에서는 사용자의 선택에 따라 서버와 클라이언트가 상호작용하여 차트를 출력하는 과정을 다음과 같은 순서로 수행한다.

1. **core / task 정보 요청**

사용자가 특정 파일을 선택하면, 클라이언트는 해당 테이블(파일)의 `core` 및 `task` 정보를 요청한다.

2. **DB 조회 및 중복 제거**

   서버는 데이터베이스에서 해당 테이블에 대해 `DISTINCT core`, `DISTINCT task` 쿼리를 실행하여 중복되지 않는 항목 목록을 조회한다.

3. **버튼 동적 생성**

   서버로부터 받은 `core` 및 `task` 정보를 바탕으로 클라이언트에서는 각각의 버튼을 동적으로 생성한다.

4. **버튼 선택 → 데이터 요청**

   사용자가 특정 `core` 또는 `task` 버튼을 클릭하면, 클라이언트는 파일 이름, 선택한 항목, 통계 기준 등을 포함하여 서버에 차트 데이터를 요청한다.

5. **서버 측 통계 쿼리 실행**

   서버는 요청된 조건에 따라 `GROUP BY` 절을 포함한 SQL 쿼리문을 실행하여, `MIN`, `MAX`, `AVG` 등의 통계 데이터를 계산한다.

6. **JSON 형태 응답 반환**

   계산된 통계 결과는 JSON 형태로 클라이언트에 반환되며, 클라이언트는 이를 파싱하여 시각화에 활용한다.

7. **Chart.js를 통한 시각화**

   클라이언트는 `Chart.js`를 활용해 선택된 데이터와 그래프 유형에 따라 차트를 실시간으로 출력한다.