

Node.js 를 활용한 Profiler 프로그램 분석



- 과 목 명 : 웹응용기술[001]
- 지도 교수 : 강영명 교수님
- 학 과 : 컴퓨터공학과
- 학 번 : 20230851
- 이 름 : 신 아 라

〈 목 차 〉

1. 프로그램 개요

- 1.1 개발 목적 및 배경
- 1.2 주요 기능 요약
- 1.3 사용 기술 및 실행 환경

2. 프로그램 수행 절차

- 2.1 사용방법 안내
- 2.2 프로그램 시작
- 2.3 데이터 입력
- 2.4 결과 출력
- 2.5 버튼 동적 생성

3. 프로그램 기능

- 3.1 Node 서버 구성
- 3.2 데이터 파일 업로드 및 파싱
- 3.3 버튼 및 차트 자동 생성
- 3.4 시각화 출력

4. 프론트엔드 시각화 구성 및 동작 방식

- 4.1 사용자 선택 기반 인터페이스 동작
- 4.2 차트 형태 선택 및 데이터 시각화
- 4.3 UI 구성 및 디자인

5. 개발 흐름과 차후 계획

- 5.1 개발 동기 및 초기 구조
- 5.2 개선 사항 및 현재 기능
- 5.3 유지 관리 및 향후 확장 가능성

1. 프로그램 개요

1.1 개발 목적 및 배경

현대의 컴퓨팅 환경에서는 멀티코어 CPU 기반의 병렬 처리가 일반화되었으며, 이에 따라 각 CPU Core의 작업 성능을 정량적으로 분석하고 시각화하는 것은 시스템의 병목 지점이나 자원 분배의 효율성을 판단하는 데 필수적인 과정이 되었다.

본 프로젝트는 사용자가 제공한 Core-Task 기반의 성능 측정 데이터를 정리하고, 각 작업 단위(core/task)에 대해 **최솟값(MIN), 최댓값(MAX), 평균(AVG), 표준편차(STD)**를 계산해 시각적 분석 도구로 제공함으로써, 실시간 데이터 분석 및 시스템 상태 진단의 기반을 마련하고자 한다.

이는 단순 수치 나열을 넘어서 시각적으로 직관적인 데이터 분석 환경을 제공함으로써, 데이터 해석력을 높이고 사용자 친화적인 분석 프로세스를 실현하는 것을 목표로 한다.

1.2 주요 기능 요약

- 사용자가 업로드한 .txt 성능 데이터 파일을 서버에서 자동 파싱하여 core-task 단위로 데이터를 분리
- 각 Core와 Task별로 MIN, MAX, AVG, STD 값을 계산
- 계산된 통계 데이터를 MySQL 데이터베이스에 저장
- 사용자가 core 또는 task를 선택하면 해당 결과를 Chart.js를 활용한 bar 또는 line 외 다양한 차트로 시각화
- 다양한 유형의 입력 파일을 지원하며, 이상치(outlier) 분석 및 표준편차 기반의 이상 데이터 탐지 가능
- 웹 상에서 파일 업로드 → 분석 → 출력까지 즉시 처리 가능한 통합형 시스템

1.3 사용 기술 및 실행 환경

백엔드 : Node.js, Express.js

프론트엔드 : EJS 템플릿 엔진, Chart.js, Bootstrap5

데이터베이스 : MySQL

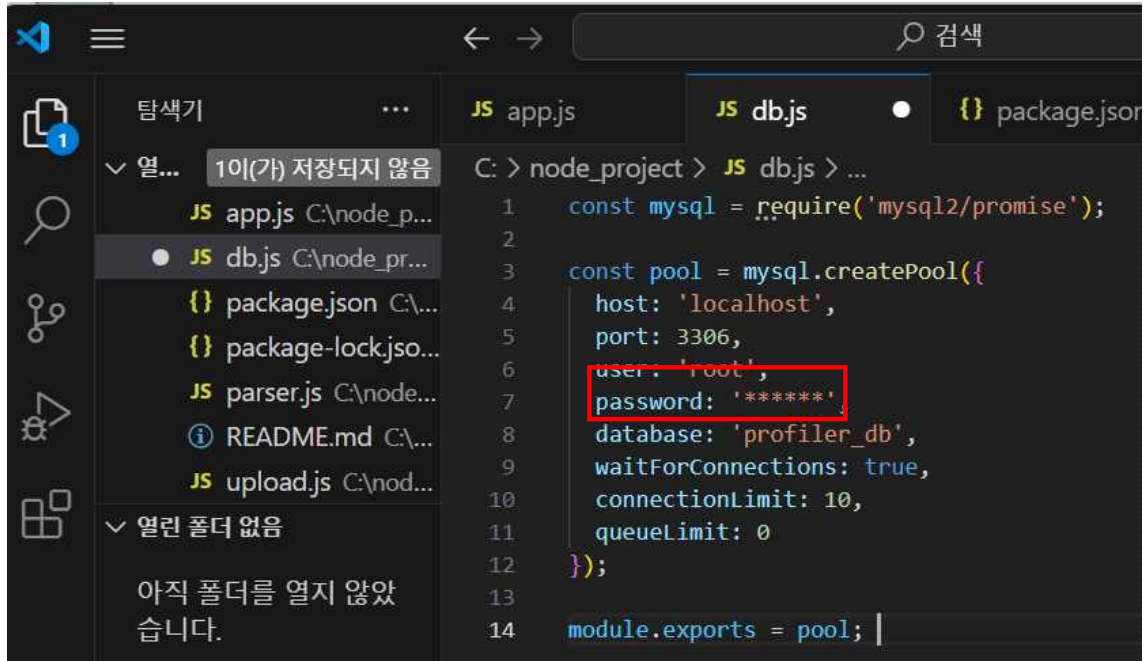
파일 업로드 처리 : multer

실행 환경

- 운영체제: Windows 11
- Node.js 버전: v22.16.0
- MySQL: Workbench 8.x 기반
- 브라우저: Chrome 기준 테스트 완료

2. 프로그램 수행 절차

2.1 사용방법 안내



[그림 1] mysql 비밀번호 수정

프로젝트의 db.js 파일로 이동하여 현재 입력되어 있는 password를 자신의 컴퓨터에 설치된 mysql의 비밀번호로 변경한다.

MySQL 데이터베이스 설정 방법

▶ 방법 1. MySQL CLI에서 직접 명령어 입력

PowerShell/CMD에서 mysql 명령어가 인식된다면 (mysql -u root -p)

→ 접속 후 다음 코드 실행

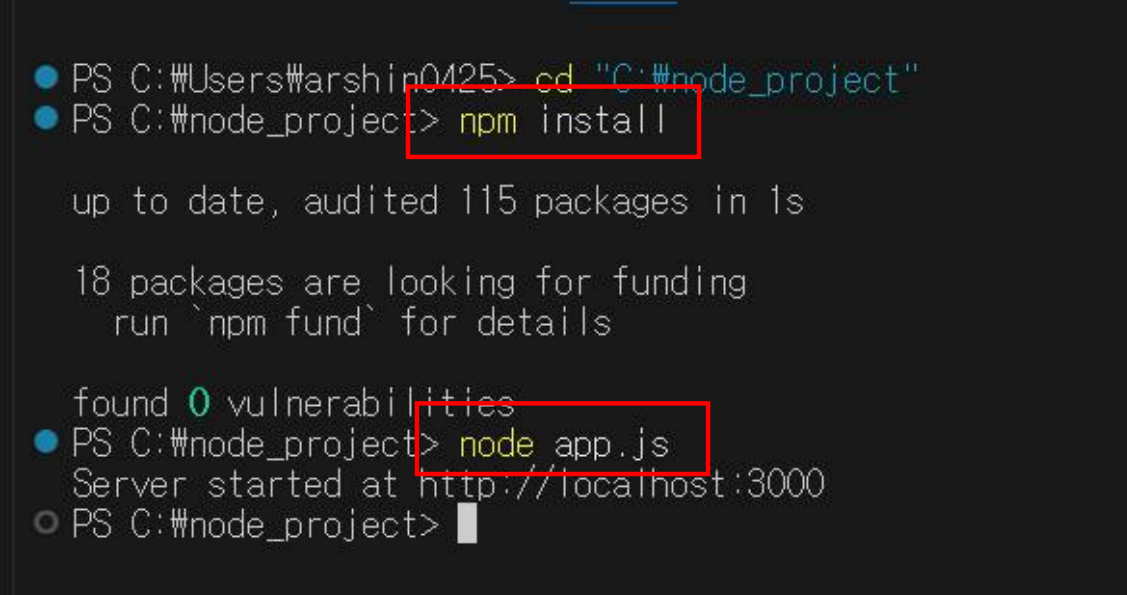
```
CREATE DATABASE profiler_db;
```

```
USE profiler_db;
```

```
CREATE TABLE profiler_data (
  id INT AUTO_INCREMENT PRIMARY KEY,
  core VARCHAR(10),
  task INT,
  min FLOAT,
  max FLOAT,
  avg FLOAT,
  std FLOAT
);
```

방법 2. 명령어가 안 된다면 → MySQL Workbench 사용

1. MySQL Workbench 실행
2. 새 SQL 탭 열기
3. 방법1에서 설명한 코드 실행

A terminal window with a dark background and light-colored text. The prompt is 'PS C:\Users\warshin0425>'. The first command is 'cd "C:\node_project"', which is followed by a new line. The second command is 'npm install', which is highlighted with a red rectangular box. The output shows 'up to date, audited 115 packages in 1s' and '18 packages are looking for funding run `npm fund` for details'. The third command is 'node app.js', which is also highlighted with a red rectangular box. The output shows 'found 0 vulnerabilities' and 'Server started at http://localhost:3000'. The prompt returns to 'PS C:\node_project>'.

```
● PS C:\Users\warshin0425> cd "C:\node_project"
● PS C:\node_project> npm install

up to date, audited 115 packages in 1s

18 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
● PS C:\node_project> node app.js
Server started at http://localhost:3000
○ PS C:\node_project>
```

[그림 2] npm 설치 및 프로젝트 실행

먼저 cd 명령어를 통해 프로젝트 디렉토리(node_project)로 이동한 후, npm install 명령어를 통해 필요한 패키지(express, mysql2, multer, ejs 등)를 node_modules 폴더에 설치한다.

패키지 설치가 완료되면, node app.js 명령어를 통해 Node.js 기반 서버를 실행한다. 정상적으로 실행되면 http://localhost:3000 주소에서 웹 서비스를 확인할 수 있으며, 이후 사용자는 웹 브라우저를 통해 텍스트 파일 업로드 및 데이터 분석을 수행할 수 있다.

2.2 프로그램 시작

```
● PS C:\Users\warshin0425> cd "C:\node_project"
● PS C:\node_project> npm install

up to date, audited 115 packages in 1s

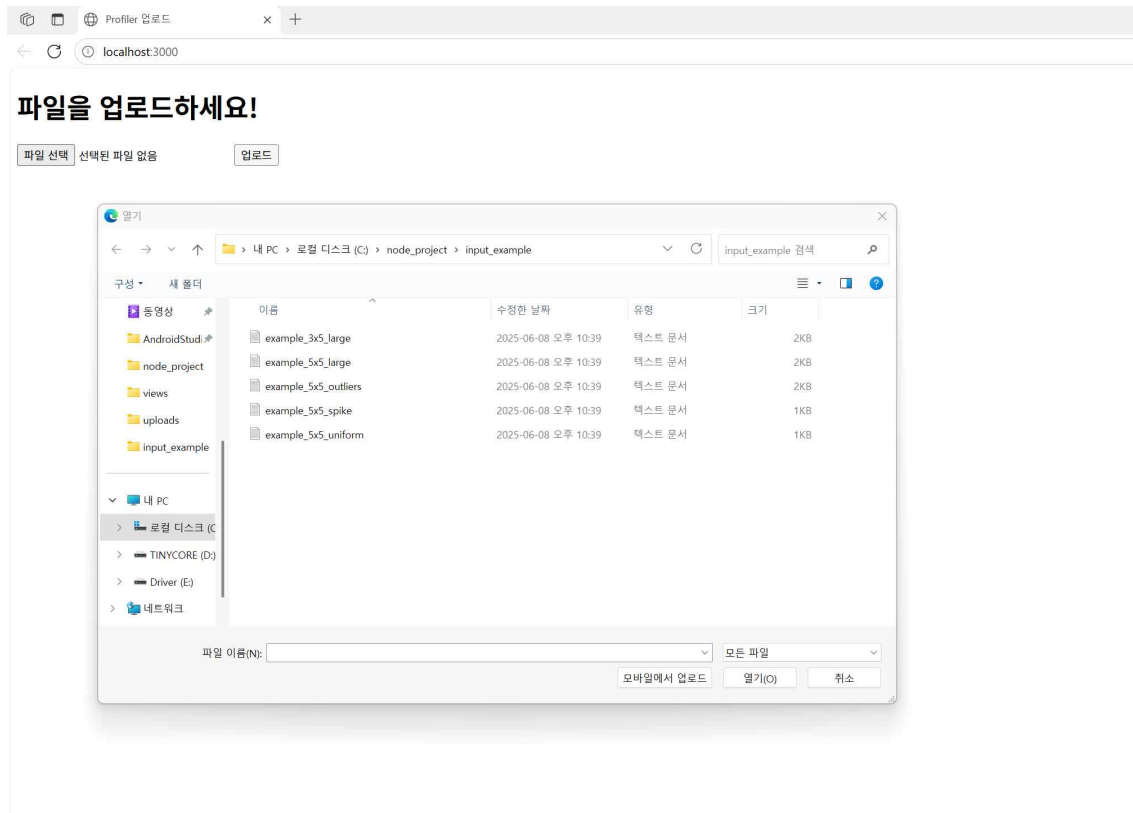
18 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
● PS C:\node_project> node app.js
  Server started at http://localhost:3000
○ PS C:\node_project> 
```

[그림 3] 주소 페이지 접속

터미널에 출력되는 주소(<http://localhost:3000>) 에 마우스 커서를 가져가서 Ctrl + Click (mac: Command + Click) 또는 커서를 대면 나타나는 링크 따라가기를 눌러 프로젝트 페이지로 접속한다.

2.3 데이터 입력



[그림 4] 파일 선택

사용자는 상단에 위치한 ‘파일 선택’ 버튼을 클릭하여 분석하고자 하는 텍스트 데이터를 선택한다. 이때 사용 가능한 입력 파일은 .txt 확장자를 가지며, 각 파일은 core별 task 성능 데이터를 포함하고 있다.

예시로는 example_3x5_large.txt, example_5x4_outliers.txt 등이 있으며, 이들은 /input_example 폴더 내에 위치한다.

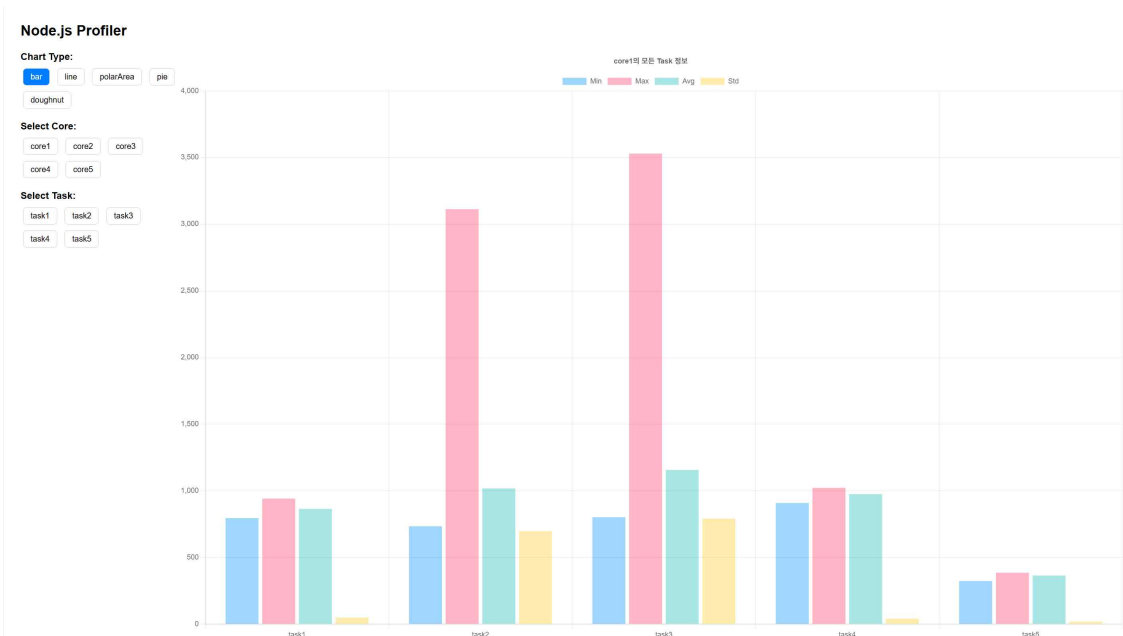
파일을 선택한 후 ‘업로드’ 버튼을 클릭하면, 해당 파일은 서버의 /uploads 디렉토리로 저장된다.

그 후 upload.js 모듈과 parser.js 모듈에서 이 파일을 읽고 데이터를 파싱한 뒤, 각 core/task별 MIN, MAX, AVG, STD 값을 계산하게 된다.

계산된 결과는 MySQL 데이터베이스(profiler_data 테이블)에 저장되며, 웹페이지에서 시각적으로 확인 가능한 상태로 전환된다.

파일이 성공적으로 처리되면, 이후에는 core 또는 task 버튼을 클릭하여 시각화 결과를 바로 확인할 수 있다.

2.4 결과 출력

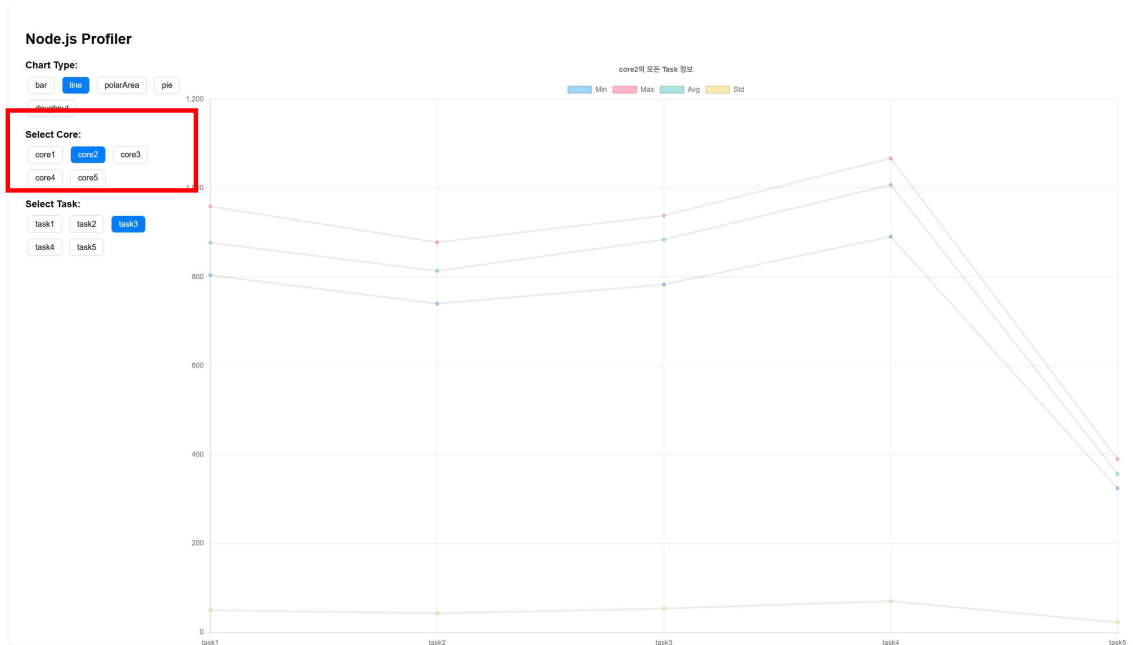


[그림 5]

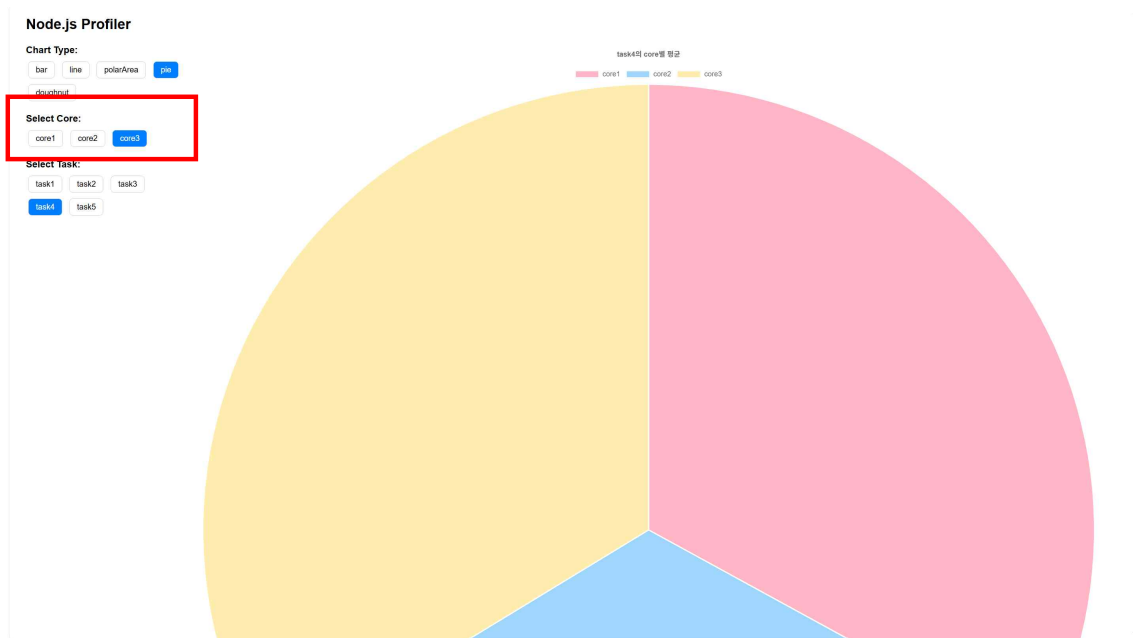
[그림 5]는 core1에 대한 모든 task 결과를 bar 차트 형태로 출력한 모습이다. 사용자는 원하는 core를 선택한 뒤, 자동으로 생성된 task 버튼 중 원하는 항목을 클릭함으로써 선택된 core에 대한 전체 task의 통계(MIN/MAX/AVG/STD)를 시각적으로 확인할 수 있다.

차트를 선택하면, 각 task별 최소값, 최대값, 평균, 표준편차가 색상별로 구분되어 표시되며, 사용자가 한눈에 값의 분포를 비교할 수 있도록 구현되었다. 시각화는 Chart.js를 통해 처리되며, 사용자 선택에 따라 실시간으로 렌더링된다.

2.5 버튼 동적 생성



[그림 6] core2의 모든 task 정보 : line 차트



[그림 7] task4의 core별 평균 : pie 차트

그림 6)과 [그림 7]은 각각 core 단위와 task 단위 선택에 따른 차트 결과를 보여준다. 특히 두 화면 모두 버튼의 개수가 입력 데이터에 따라 자동으로 동적으로 생성되는 구조임을 확인할 수 있다.

본 프로젝트에서는 사용자가 업로드한 프로파일링 텍스트 파일(inputFile.txt)을 서버에서 파싱한 후, core와 task의 개수를 동적으로 판단하여 HTML 내 버튼을 자동 생성한다. 예를 들어, [그림 6]은 5개의 core와 5개의 task가 포함된 데이터가 입력되었을 경우를 보여주며, 그에 따라 core 버튼은 5개, task 버튼도 5개가 생성된 상태이다.

반면 [그림 7]은 core가 3개뿐인 데이터셋을 기준으로 출력된 결과로, core 버튼은 3개만 생성되어 있다. 이처럼 버튼 개수는 서버에서 분석된 결과(result JSON 객체)를 기반으로 EJS 템플릿 내 반복문(<% for %>, <% Object.keys().forEach() %>)을 통해 자동 생성된다.

이를 통해 사용자는 입력 데이터의 크기와 상관없이 유연하게 UI를 사용할 수 있으며, 매번 수동으로 버튼을 생성하거나 수정할 필요 없이 자동화된 인터페이스를 통해 다양한 입력 파일을 시각적으로 분석할 수 있다.

또한, 차트 형태 역시 버튼 클릭으로 즉시 변경 가능하며, Chart.js 라이브러리의 구조를 활용하여 그래프 유형(line, bar, pie 등)에 따라 각기 다른 시각화 방식으로 렌더링된다.

3. 프로그램 기능

3.1 Node 서버 구성

본 프로젝트는 Node.js 환경에서 실행되는 웹 기반 프로파일링 도구로, 사용자가 업로드한 데이터 파일을 분석하여 통계 정보를 시각화하는 기능을 제공한다. 서버 구동은 루트 디렉토리에 위치한 app.js 파일을 실행함으로써 이루어진다.

- 서버 주요 구성 요소
 - 다음은 프로젝트의 주요 디렉토리 및 파일과 그 역할이다

/app.js	웹 서버 실행의 진입점. express를 통해 라우팅, ejs 뷰 렌더링, MySQL 연결 및 모듈 호출을 수행
/db.js	MySQL 연결을 정의하고 내보내는 설정 파일
/upload.js	multer 모듈을 통해 파일 업로드를 처리하고 parser 호출을 담당하는 라우팅 모듈
/parser.js	업로드된 텍스트 파일을 읽어 각 core/task에 대해 MIN, MAX, AVG, STD를 계산하고 DB에 저장하는 핵심 분석 로직
/views/	EJS 기반의 템플릿 뷰 파일들을 저장하는 디렉토리 (업로드 화면, 결과 차트 화면 포함)
/node_modules/	npm install을 통해 설치된 외부 라이브러리 디렉토리 (express, multer, chart.js 등 포함)

- 주요 라이브러리
 - express: Node 서버 구축 및 라우팅 처리
 - ejs: 템플릿 렌더링 엔진
 - multer: 파일 업로드 처리
 - mysql2: MySQL 연동
 - chart.js: 클라이언트 측 데이터 시각화

- 실행 방식
 - 서버는 node app.js 명령어로 실행된다.
 - 로컬 브라우저에서 http://localhost:3000으로 접속하면 업로드 화면이 나타난다.
 - 업로드 후 분석 결과는 /upload 경로에서 시각화된 형태로 확인 가능하다.

3.2 데이터 파일 업로드 및 파싱

사용자는 /upload 경로에서 분석하고자 하는 텍스트 파일을 업로드할 수 있다. 업로드 가능한 파일은 기본적으로 inputFile.txt이지만, 테스트 및 확장 목적을 위해 프로젝트 내의 /input_example 폴더에 존재하는 다수의 예제 파일들도 선택 가능하다. 예를 들어, example_3x5_large.txt, example_5x5_outliers.txt와 같은 파일들을 직접 선택해 분석할 수 있다.

업로드된 파일은 multer 미들웨어를 통해 /uploads 디렉토리에 저장되며, 서버는 이후 이 파일을 기반으로 프로파일링 분석을 수행한다.

- 처리 흐름

1. 파일 업로드

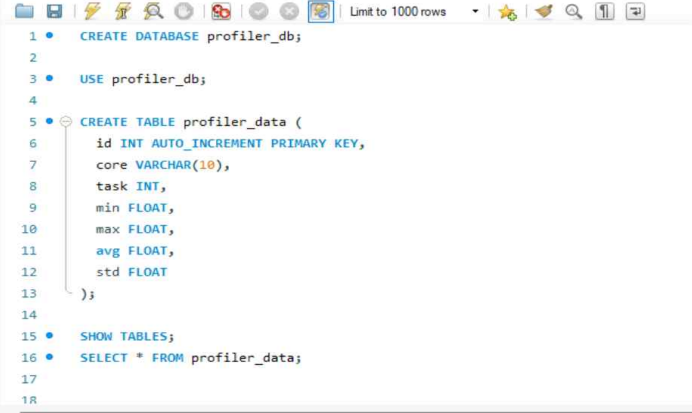
사용자는 HTML `<input type="file">` 요소를 통해 로컬 파일을 선택

선택된 파일은 서버 측에서 /uploads 폴더에 저장됨

2. 파일 파싱 (parser.js)

parser.js는 업로드된 파일을 행 단위로 읽어 core-task 구조의 이차원 배열로 파싱

각 task 데이터에 대해 다음 통계값을 계산: MIN, MAX, AVG, STD (표준편차)



The screenshot shows a database management interface with a SQL editor and a result grid. The SQL editor contains the following queries:

```
1 • CREATE DATABASE profiler_db;
2
3 • USE profiler_db;
4
5 • CREATE TABLE profiler_data (
6   id INT AUTO_INCREMENT PRIMARY KEY,
7   core VARCHAR(10),
8   task INT,
9   min FLOAT,
10  max FLOAT,
11  avg FLOAT,
12  std FLOAT
13 );
14
15 • SHOW TABLES;
16 • SELECT * FROM profiler_data;
17
18
```

The result grid displays the data for the profiler_data table, showing 19 rows of data. The columns are id, core, task, min, max, avg, and std.

id	core	task	min	max	avg	std
1	core1	1	796	942	864.80	49.99
2	core1	2	734	3113	1018.70	698.96
3	core1	3	803	3530	1157.20	792.03
4	core1	4	909	1023	975.80	41.18
5	core1	5	324	386	365.00	18.60
6	core2	1	804	959	877.20	50.03
7	core2	2	740	878	813.70	42.80
8	core2	3	783	938	884.20	53.56
9	core2	4	891	1067	1007.50	70.20
10	core2	5	324	390	356.70	22.42
11	core3	1	815	2862	1076.20	596.88
12	core3	2	761	877	828.00	43.66
13	core3	3	795	945	868.70	50.76
14	core3	4	901	1064	985.20	53.68
15	core3	5	336	773	408.00	122.69
16	core4	1	820	937	884.30	37.44
17	core4	2	744	1715	908.20	272.66
18	core4	3	782	906	845.60	48.26
19	core4	4	908	1057	958.20	40.75

[그림 8] profiler_data 테이블에 저장된 분석 결과 예시

3. DB 저장 (MySQL)

계산된 통계값은 profiler_data 테이블에 삽입됨

컬럼: core, task, min, max, avg, std

데이터 삽입은 core별, task별로 반복 수행됨

4. 시각화 준비

분석 결과가 DB에 저장되면, 클라이언트는 이를 조회하여 차트로 시각화할 수 있음

사용자는 core, task를 선택하여 Chart.js 기반의 다양한 형태(bar, line, pie 등)로 분석 결과를 확인할 수 있음

3.3 버튼 및 차트 자동 생성

사용자가 업로드한 파일의 데이터 구조에 따라 core 수, task 수가 동적으로 계산되며, 이에 따라 화면 상에 버튼이 자동 생성된다. 이 동작은 서버에서 전달된 분석 결과 JSON(result)을 기반으로 views/result.html 내 EJS 반복문으로 처리된다.

- core 버튼: core1, core2, ..., coreN

- task 버튼: task1, task2, ..., taskM

버튼 클릭 시 JavaScript에서 core 또는 task 기준으로 데이터를 선택하여 차트를 다시 그린다.

3.4 시각화 출력 (Chart.js)

차트 출력은 Chart.js를 활용하여 구현된다. 사용자가 선택한 chart type (bar, line, polarArea, pie, doughnut)에 따라 적절한 형태로 데이터를 시각화한다.

core 기준 출력: 선택된 core에 대해 각 task별 min/max/avg/std 값을 차트로 출력

task 기준 출력: 선택된 task에 대해 각 core별 평균(avg) 값을 출력

Chart type 선택: 버튼 클릭으로 실시간 변경 가능 (bar → line 등)

canvas 크기 최적화: 버튼/차트 영역을 분할하여 화면을 넓게 활용

4. 프론트엔드 시각화 구성 및 동작 방식

4.1 사용자 선택 기반 인터페이스 동작

데이터 분석이 완료되면 서버는 각 core와 task에 대한 통계 정보를 JSON 형식으로 클라이언트에 전달한다. 클라이언트는 이를 기반으로 사용 가능한 core 수와 task 수를 자동으로 판단하여 버튼을 동적으로 생성한다.

core 버튼은 core1, core2, ... 형식으로 생성되며, core별 모든 task 통계를 탐색할 수 있다.

task 버튼은 task1, task2, ... 형식으로 생성되며, task별 각 core의 비교 분석이 가능하다. 버튼은 <button> 태그로 생성되며, 선택 시 .active 클래스를 적용하여 시각적인 피드백을 제공한다. 이 구조는 입력 파일 구조가 달라지더라도 UI를 자동으로 적응시킬 수 있게 한다.

4.2 차트 형태 선택 및 데이터 시각화

차트는 Chart.js 라이브러리를 이용하여 렌더링되며, 시각화 형태는 다음 중 하나를 선택할 수 있다 : Bar Chart (기본값), Line Chart, Polar Area, Pie Chart, Doughnut

차트 유형은 버튼으로 선택되며, 선택 즉시 기존 차트를 destroy()한 후 새 차트를 다시 그리는 방식으로 동작한다.

선택된 core 또는 task에 따라 서버에서 전달받은 min, max, avg, std 값들을 차트에 맞게 가공하여 표시한다.

- core 기준 선택 시 → 한 core의 모든 task 통계
- task 기준 선택 시 → 하나의 task에 대한 모든 core 통계

각 차트의 타이틀은 자동으로 설정되어 어떤 데이터를 시각화하고 있는지 명확히 보여준다.

4.3 UI 구성 및 디자인

버튼과 전체 레이아웃은 Bootstrap 5의 기본 컴포넌트를 활용해 구성되었다. 시각적으로 깔끔하고 일관성 있는 버튼 디자인(.btn, .active)을 적용하였고, 반응형 특성을 고려하여 기본적인 여백과 정렬(margin, padding)도 포함하였다.

캔버스 영역은 시각화 집중도를 높이기 위해 좌측 메뉴 영역과 분리되어 넓은 공간에 차트를 배치할 수 있도록 설정되었다. 이를 통해 대규모 데이터셋도 한눈에 파악 가능하게 구현하였다.

5. 개발 흐름과 차후 계획

5.1 개발 동기 및 초기 구조

본 프로젝트는 단일 텍스트 파일(inputFile.txt)을 업로드하여, CPU의 core와 각 task별 성능 데이터를 수집하고 통계를 계산한 뒤 시각화하는 것을 목적으로 시작되었다.

초기 버전에서는 MIN, MAX, AVG 통계값만 계산되었으며, 결과 출력도 서버 콘솔을 통해 이루어졌기 때문에 사용자는 데이터를 직관적으로 파악하기 어려웠고, 시각화 기능이나 데이터 재활용이 제한적이었다.

5.2 개선 사항 및 현재 기능

기능 개선과 구조적 확장을 위해 다음과 같은 변화가 이루어졌다.

- 업로드 → 분석 → DB 저장 → 시각화로 이어지는 전반적 흐름을 완성
- 표준편차(STD) 추가 계산 및 MySQL 테이블(profiler_data)에 함께 저장
- 동일 테이블 중복 생성을 방지하는 파일명 체크 및 자동 테이블 생성 기능 구현
- Chart.js를 이용한 다양한 시각화 차트 지원 (bar, line, pie, doughnut, polarArea 등)
- 입력 파일의 구조에 따라 core, task 선택 버튼이 동적으로 생성되도록 구성
- 프론트엔드에서 시각적으로 비교 가능한 차트를 통해 분석 결과의 가독성 향상

5.3 유지 관리 및 향후 확장 가능성

현재 버전은 단일 파일 업로드를 기반으로 동작하지만, 추후 다음과 같은 확장이 가능하다.

- 다중 파일 업로드 또는 디렉토리 단위 자동 파싱 기능으로 확장
- 추가 통계 지표(Median, IQR 등) 도입을 통한 분석 정확도 향상
- 시각화 UI에서 데이터 필터, 범례 설정, 차트 다운로드 기능 등의 인터랙션 요소 추가
- 데이터 삭제, 초기화 기능 도입을 통해 반복 사용성을 높임
- 웹페이지 인터페이스 상에서 업로드 내역, 분석 이력 관리 페이지 구현 가능성 있음