REPORT

성결대학교 SUNGKYUL UNIVERSITY

과목명		웹응용기술
담당교수		강영명 교수님
학과		컴퓨터 공학과
학년		3학년
학번		20211034
이름		황예림
제출일		2025-06-06

< 목차 >

1. 프로그램 개요

1-1 개발 목적

이번 프로그램은 성능 프로파일링 데이터를 기반으로 효율적인 데이터 분석 및 시각화를 제공하기 위해 개발되었다. 기존에 제공된 코드 기반의 기능을 분석하고 이를 개선하여, 파일 업로드, 통계 계산, 시각화 기능을 통합한 하나의 프로그램을 구축하였다. 이를 통해 사용자는 브라우저를 통해 텍스트 형식의 데이터를 간편하게 업로드하고, 서버에서 처리된 분석 결과를 다양한 차트 형태로 직관적으로 확인할 수 있다. 또한, 단순한 데이터 처리에 그치지 않고, 실제 활용 가능한 프로그램 형태로 확장하는 것을 목표로 하였다.

1-2 주요 기능 및 사용 기술

구분	내용
파일 업로드	브라우저를 통한 inputFile.txt 업로드 기능
데이터 분석	업로드된 데이터를 기반으로 MIN, MAX, AVG, STD 계산
DB 저장	Sequelize ORM을 이용해 분석 데이터를 SQLite DB에 저장
시각화	Chart.js를 이용한 Bar, Line, Pie 차트 시각화
사용 기술	Node.js, Express, Sequelize, EJS, Chart.js, Multer 등

표 1) 주요 기능 및 사용기술 정리

2. 프로그램 사용 및 수행 절차

2-1 전체 시스템 구성 요약

본 프로그램은 Node.js 기반의 백엔드 서버, EJS를 이용한 프론트엔드 인터페이스, Sequelize ORM을 통한 데이터베이스 연동, 그리고 Chart.js를 활용한 시각화 기능으로 구성되어 있다. 사용자가 브라우저를 통해 inputFile.txt 파일을 업로드하면 서버는 해당 파일을 분석하여 통계 정보를 산출한 뒤 데이터베이스에 저장하고, 그 결과를 시각화하여 웹 화면에 출력한다. 전체 프로그램의 동작 흐름은 다음과 같다.

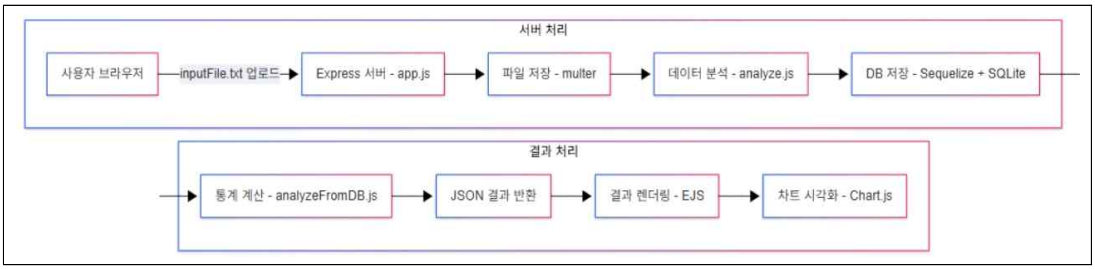


그림 1) 프로그램 전체 Flow Chart

사용자는 웹 페이지에 접속하여 inputFile.txt 파일을 선택하고 '업로드 및 분석' 버튼을 클릭한다. 업로드 요청은 Express 서버(app.js)로 전달되며, 서버는 multer 미들웨어를 통해 업로드된 파일을 서버의 지정된 경로에 저장한다. 저장된 파일은 analyze.js 모듈에서 읽어 들여 각 Core와 Task 단위로 데이터를 파싱한다. 파싱된 데이터는 Sequelize ORM을 이용하여 SQLite 데이터베이스의 Record 테이블에 저장된다. 이후 analyzeFromDB.js 모듈이 해당 데이터를 조회하고, Task별 통계를 계산한다. 이 분석 결과는 JSON 형태로 클라이언트에 전달되며, EJS 템플릿이 결과를 HTML 페이지로 렌더링하고, 최종적으로 Chart.js를 통해 Bar, Line, Pie 차트로 시각화된다.

2-2 주요 기능 흐름

본 프로그램은 사용자의 파일 업로드부터 결과 시각화까지 일련의 흐름을 따라 동작한다. 각 단계별 주요 기능과 흐름은 다음과 같다.

1) 파일 업로드 및 서버 저장

그림 2) 파일 업로드 화면

사용자는 웹 브라우저를 통해 .txt 형식의 입력 파일을 업로드한다. 이 파일은 <input type="file"> 요소와 POST /upload 요청을 통해 서버로 전달되며, multer

미들웨어를 이용해 서버의 지정된 디렉터리(/uploads)에 저장된다.

2) 데이터 파싱 및 가공

	task1	task2	task3	task4	task5
core1	886	749	849	909	352
core2	959	849	788	1053	324
core3	942	867	930	1064	365
core4	820	817	816	929	336
core5	803	786	929	958	329

그림 3) 입력 데이터 형식 예시 - inputFile.txt

저장된 텍스트 파일은 analyze.js 모듈에서 읽혀진다. 이 모듈은 각 행을 Core별로 나누고, 각 열을 Task별로 구분하여 숫자 데이터를 추출한다. 추출된 데이터는 Record 모델을 통해 SQLite DB에 저장된다.

3) 통계 정보 계산

업로드된 데이터를 기반으로 analyzeFromDB.js는 Record 테이블을 조회하여 각 Task별로 최소값(MIN), 최대값(MAX), 평균(AVG), 표준편차(STD)를 계산한다. 추가적으로 Core별 통계도 함께 산출하여 비교 분석이 가능하도록 구성하였다.

4) 결과 전달 및 페이지 렌더링

서버에서 분석한 통계 데이터는 JSON 형태로 EJS 템플릿에 전달되며, 결과 페이지는 upload.ejs에서 렌더링된다. 사용자 브라우저에 시각화된 결과가 포함된 HTML 페이지가 출력된다.

5) 차트 시각화

클라이언트 측에서는 Chart.js를 이용해 Bar, Line, Pie 차트로 통계 결과를 확인할 수 있다. 또한, Core별 Task 분석도 Line Chart로 표현되며, 사용자는 각 통계 지표(MIN, MAX, AVG, STD)를 선택하여 시각화 항목을 제어할 수 있다.

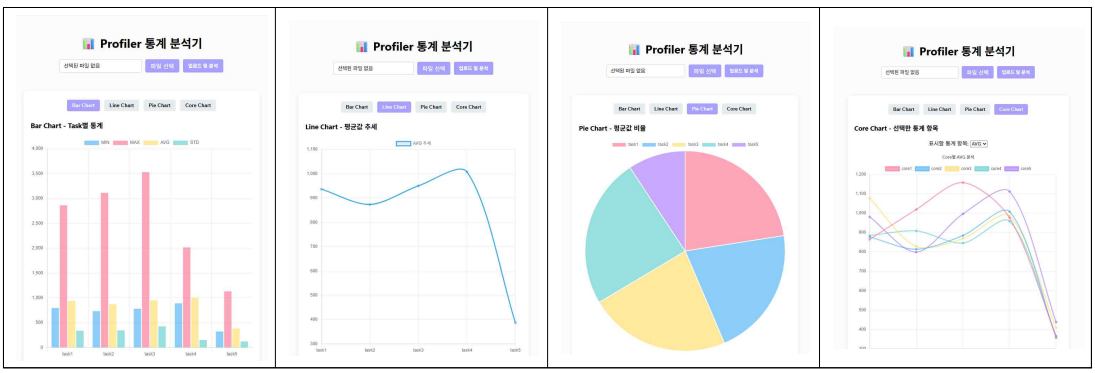


그림 4) 시각화 결과 화면 - Bar, Line, Pie, Core Chart

3. 주요 기능 및 코드 설명

3-1 프로젝트 구조 및 사용 라이브러리

1) 주요 라이브러리 및 역할

구분	내용
백엔드 프레임워크	Express (서버 라우팅, 요청 처리)
파일 업로드	Multer (multipart/form-data 처리)
ORM	Sequelize (SQLite DB 연동 및 모델 정의)
프론트 템플릿	EJS (결과 페이지 렌더링)
차트 시각화	Chart.js (Bar, Line, Pie, Core Chart 등 시각화)
기타	nodemon (개발 편의), path, fs 등 기본 모듈

표 2) 주요 라이브러리

2) 프로젝트 폴더 및 주요 파일 구조

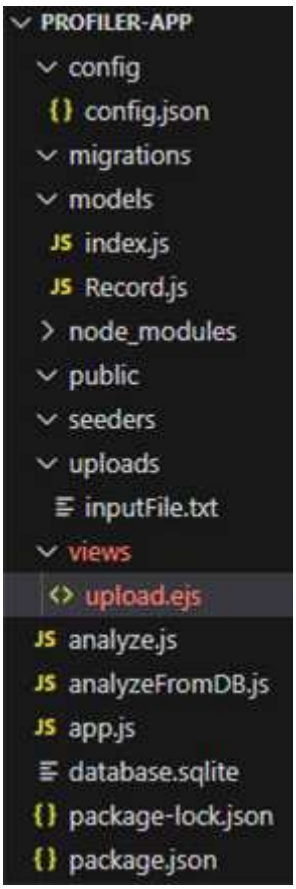
	**/config/config.json** : Sequelize 설정 파일 (SQLite DB 연결 정보 포함)
	/models/Record.js : Sequelize 기반 데이터 모델 정의. 각 Core/Task의 분석 데이터를 저장
	/uploads/inputFile.txt : 사용자가 업로드한 입력 텍스트 파일 저장 위치
	/views/upload.ejs : 업로드 및 결과 화면을 구성하는 EJS 템플릿 파일
	analyze.js : 업로드된 텍스트 파일을 파싱하여 DB에 저장하는 분석 로직
	analyzeFromDB.js : DB에서 데이터를 조회하고 MIN, MAX, AVG, STD 값을 계산
	app.js : 전체 서버 구동, 라우팅 처리, 템플릿 렌더링을 담당하는 메인 서버 파일

표 3) 프로젝트 디렉토리 구조

3-2 파일 업로드 처리 방식 (app.js, multer)

사용자는 웹 브라우저를 통해 .txt 파일을 업로드하며, 서버는 이를 multer 모듈을 통해 /uploads 디렉토리에 저장한다. 이후 해당 파일은 analyze.js 모듈에서 분석되어 데이터베이스에 저장된다.

- 클라이언트에서 <input type="file"> 요소로 선택된 파일은 POST /upload 요청을 통해 서버에 전송된다.
- Express 서버는 multer를 통해 파일을 저장하며, 업로드된 파일의 이름은 고정(inputFile.txt)으로 지정된다.
- 저장이 완료되면 analyze.js 모듈이 호출되어 파일을 읽고 파싱한 후, 데이터베이스에 저장하는 처리를 수행한다.

3-3 데이터 분석 및 통계 처리 (analyze.js, analyzeFromDB.js)

업로드된 입력 파일은 analyze.js에서 파싱되어 데이터베이스에 저장되며, analyzeFromDB.js는 저장된 데이터를 조회하여 통계값(MIN, MAX, AVG, STD)을 계산한다.

- 입력 파일은 Core 단위로 행을 구분하고, 각 행은 Task 단위로 열이 구분된 형식이다.
- 각 Core/Task 조합의 값은 Sequelize ORM을 이용하여 Record 테이블에 저장된다.
- 매 업로드 시 기존 데이터는 삭제(destroy) 후 새롭게 저장되어 중복 저장을 방지한다.
- analyzeFromDB.js는 Task별, Core별로 데이터를 그룹화하여 통계값을 계산한다.
- 최종 분석 결과는 JSON 형태로 클라이언트에 전달되며, 이후 시각화에 사용된다.

3-4 데이터베이스 모델 구성 (models/record.js)

- Record 모델은 Sequelize를 통해 정의되며, 주요 필드는 core, task, value이다.
- 분석된 모든 데이터는 SQLite 기반의 단일 Record 테이블에 저장된다.
- Sequelize ORM을 활용하면 SQL 문법 없이도 데이터를 조작할 수 있어 코드의 가독성과 유지보수성이 향상된다.

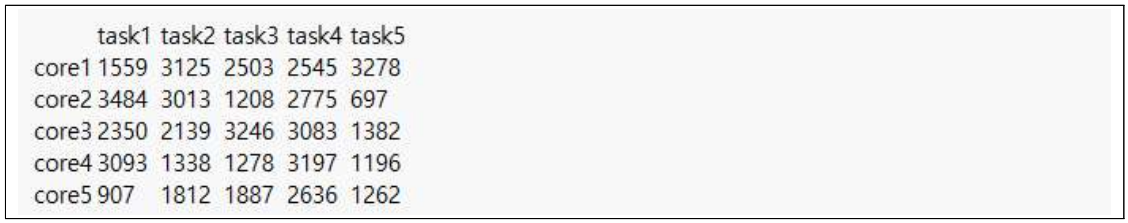
3-5 결과 페이지 구성 및 시각화 처리 (upload.ejs + Chart.js)

- 분석 결과는 서버에서 upload.ejs 템플릿에 JSON 형태로 전달된다.
- 클라이언트는 Chart.js를 이용해 Bar, Line, Pie 차트를 시각화하며, Core별 Task 분석도 Line Chart로 렌더링된다.
- 사용자는 체크박스를 통해 MIN, MAX, AVG, STD 항목을 선택하여 시각화 항목을 동적으로 제어할 수 있다.

4. 실행 결과 및 테스트 예시

4-1 샘플 입력 파일을 통한 테스트

테스트를 위해 3000개의 데이터를 포함한 대용량 입력 파일 inputFile_3000.txt를 직접 생성하였다. 각 Core는 task1부터 task5까지 5개의 수치 데이터를 포함하며, 값은 0부터 5000 사이의 무작위 값으로 설정되어 있어 다양한 통계 패턴을 시각적으로 확인할 수 있도록 설계되었다. 해당 파일은 다음과 같은 구조로 구성되어 있다.



	task1	task2	task3	task4	task5
core1	1559	3125	2503	2545	3278
core2	3484	3013	1208	2775	697
core3	2350	2139	3246	3083	1382
core4	3093	1338	1278	3197	1196
core5	907	1812	1887	2636	1262

그림 5) 입력 데이터 예시 (inputFile_3000.txt)

- 각 행은 Core 단위를, 각 열은 Task 단위를 나타낸다.
- 각 5개의 Core와 5개의 Task 값으로 구성된 총 25개의 값이 한 세트로 Record 테이블에 저장된다.

4-2 시각화 결과 페이지 예시 및 설명

3000개의 데이터를 기반으로 분석을 수행한 결과, 다음과 같은 시각화 화면이 Chart.js를 통해 출력된다. 모든 그래프는 사용자의 선택에 따라 동적으로 갱신되며, 다양한 통계 정보를 직관적으로 확인할 수 있다.

- Bar Chart
  → 각 Task별로 MIN, MAX, AVG, STD를 막대 그래프로 출력
  → 데이터 비교가 직관적으로 가능함

- Line Chart
  → Task별 평균값의 추세를 선형 그래프로 시각화
  → 값의 증감 패턴을 파악 가능

- Pie Chart
  → Task별 평균값의 비율을 원형 그래프로 시각화
  → 각 Task가 전체 데이터에서 차지하는 비중을 직관적으로 확인 가능

- Core Chart
  → Core별 Task 변화 추세를 선형 그래프로 표현
  → 사용자 선택에 따라 MIN, MAX, AVG, STD 항목 전환 가능

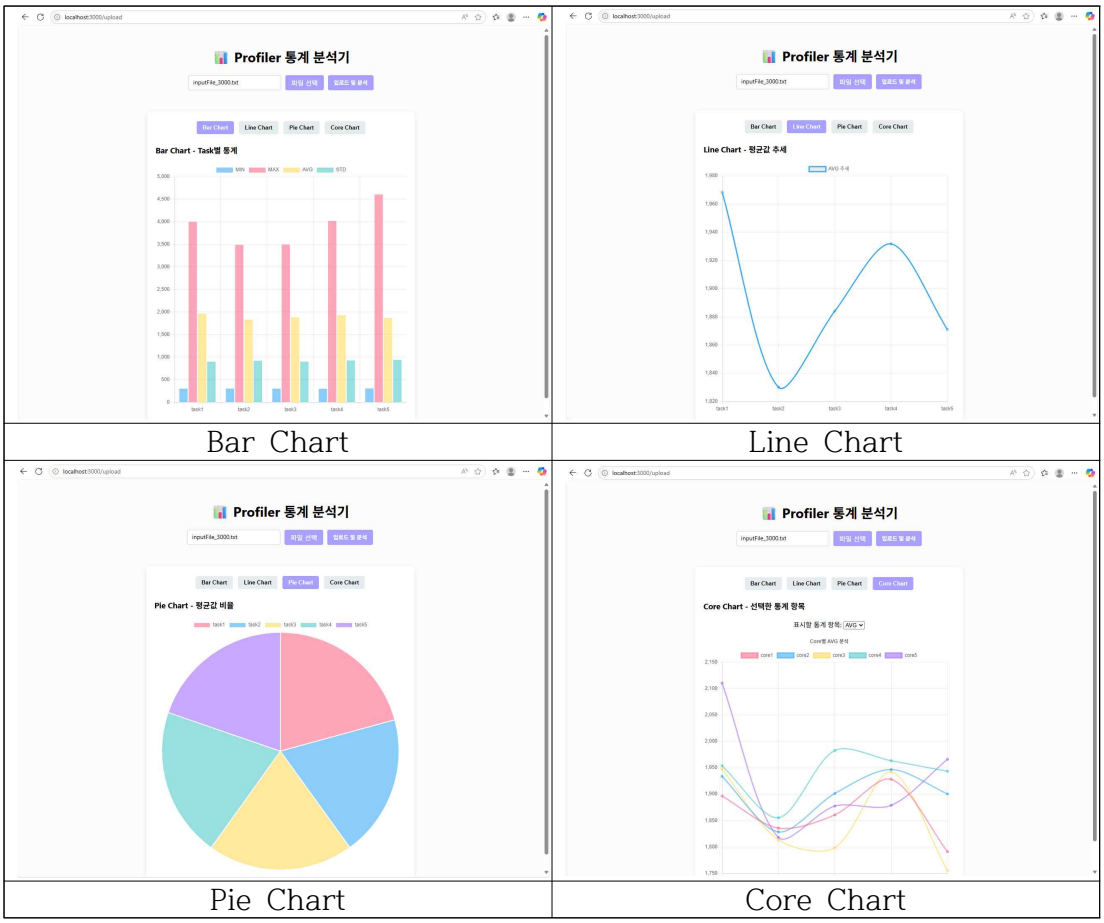


그림 6) 차트 시각화 결과

4-3 주요 기능별 정상 작동 여부 확인

다음 표는 주요 기능들에 대해 inputFile_3000.txt를 기반으로 테스트한 결과를 요약한 것이다. 대용량 데이터 처리, 통계 분석, 시각화 렌더링까지 모든 기능이 정상적으로 동작함을 확인하였다.

테스트 항목	결과	설명
대용량 데이터 업로드	성공	3000개 데이터 약간의 로딩 후 정상 업로드
데이터 파싱 및 DB 저장	성공	모든 데이터가 누락 없이 DB에 삽입됨
통계 계산 (MIN, MAX, AVG, STD)	성공	Core별, Task별 통계 산출 정확
시각화 렌더링 (Chart.js)	성공	모든 그래프가 즉시 반영되며 렉 없이 출력
사용자 조작 기능	성공	통계 항목 드롭다운/탭 전환 모두 정상 작동

표 4) 주요 기능 및 사용기술 정리

5. 결론

5-1 프로젝트 요약 및 배운 점

이번 프로젝트는 Node.js 기반의 웹 애플리케이션을 개발하여 대용량 입력 파일을 업로드하고, 이를 분석해 시각화하는 전체 프로세스를 구현한 작업이었다. 사용자는 텍스트 파일을 업로드하고, 서버는 multer 모듈로 파일을 저장한 뒤, 해당 파일을 파싱하여 데이터베이스에 기록한다. 이후 Chart.js를 활용해 다양한 통계 지표(MIN, MAX, AVG, STD)를 시각화함으로써 실시간 분석 결과를 사용자에게 제공할 수 있었다. 이 프로젝트를 통해 백엔드 개발의 기본 구조를 이해하고, 입력 데이터의 구조 설계, ORM을 통한 데이터 처리, 클라이언트-서버 간의 데이터 흐름, 그리고 Chart.js를 통한 시각화 방법까지 전반적인 웹 서비스 개발 프로세스를 경험할 수 있었다. 특히 실제 입력 데이터를 기반으로 동작하는 시스템을 구축하면서 입력-처리-분석-출력의 흐름을 명확히 체득할 수 있었다.

5-2 향후 개선 방향 및 확장 가능성

현재 구현된 시스템은 하나의 정적인 파일 업로드-분석 흐름으로 구성되어 있으나, 향후 다음과 같은 확장이 고려될 수 있다.

- 입력 형식 확장: task 개수나 core 수가 유동적인 입력 파일도 지원하도록 개선 (예: dynamic table parser)
- 데이터 처리 속도 개선: 업로드한 데이터의 개수가 더 커져도 빠르게 처리할 수 있도록 속도 개선
- 비교 기능 추가: 이전에 업로드한 데이터와 새 데이터의 통계 비교 기능 구현
- 데이터 필터링 및 검색: 특정 task나 core에 대한 필터 기능 추가로 사용자 맞춤 분석 제공
- 반응형 UI 및 시각화 강화: Chart.js 외에 D3.js 등 다양한 라이브러리를 적용하여 더욱 풍부한 시각화를 구현
- 다중 사용자 환경 지원: 로그인 기능을 추가하여 사용자별 업로드 및 분석 이력 관리 가능
- 배포 및 서비스화: 클라우드 환경에 배포하여 실제 서비스로 활용 가능 (예: AWS, Vercel 등)

이러한 개선을 통해 현재의 분석 도구를 더욱 실용적이고, 다양한 목적에 활용할 수 있는 웹 기반 플랫폼으로 확장할 수 있을 것이다.