# REPORT

---

## Node.js 를 활용한

## Profiler 프로그램 분석 보고서


20211373 김가을

# <목차>



# <그림 목차>

# 1. 프로그램 수행 절차 분석

## 1.1 사용 방법 안내

1. 비밀번호 수정
   프로젝트를 로컬 환경에서 실행하려면 `javaweb/config/config.json` 파일에 있는 MySQL 비밀번호를 로컬 MySQL 서버의 비밀번호로 변경해야 합니다. 이는 데이터베이스에 접근하기 위한 인증 절차입니다.
2. DB 수정 및 선택
   터미널에서 `root` 계정으로 MySQL에 접속한 후 `use javaweb;` 명령어를 통해 데이터베이스를 생성하고 사용합니다. 콘솔에 출력된 내용을 확인한 후 `exit` 명령어로 MySQL을 종료합니다. 이는 데이터 저장을 위한 데이터베이스 환경을 설정하는 과정입니다.
3. 프로젝트 설치 및 실행
   프로젝트의 루트 디렉토리에서 `npm install` 명령어를 실행하여 필요한 Node.js 패키지를 설치합니다. 그런 다음, `npm start` 명령어로 서버를 시작합니다. 이 과정은 프로젝트가 의존하는 모든 패키지를 설치하고 서버를 시작하여 프로젝트를 로컬 환경에서 실행 가능하게 합니다.
4. 프로젝트 시작
   초기 설정이 완료되면, 터미널에 표시된 URL을 웹 브라우저에 입력하여 프로젝트에 접속합니다. 여기서 프로젝트의 기능을 테스트하고 사용해 볼 수 있습니다.

## 1.2 데이터 입력

1. 파일 선택
   데이터 입력 화면에서 `파일 선택` 버튼을 클릭하여 업로드할 데이터 파일을 선택합니다. 이 기능은 사용자가 파일 시스템에서 특정 데이터를 서버로 업로드할 수 있습니다.
2. 데이터 입력 처리
   파일을 선택한 후 제출하면, 서버는 파일을 읽고 데이터베이스에 저장합니다. 파일 내 숫자가 아닌 문자가 포함된 데이터는 무시됩니다. 이 단계에서 데이터 유효성 검사를 통해 올바른 형식의 데이터만 저장됩니다.

## 1.3 결과 출력

1. 차트 생성
   데이터가 정상적으로 저장되면, 데이터 리스트에서 특정 데이터를 선택하고 차트 유형을 선택하여 결과를 시각화합니다. 사용자는 `Core`와 `Task`를 선택하여 원하는 데이터를 차트로 확인할 수 있습니다.
2. 버튼 동적 생성

데이터베이스에 저장된 데이터의 `Core`와 `Task` 개수에 따라 동적으로 버튼을 생성하여 사용자가 쉽게 데이터를 선택하고 차트를 생성할 수 있습니다.

**1.4** 데이터 삭제

필요 없는 데이터는 데이터베이스 리스트에서 삭제 버튼을 클릭하여 제거할 수 있습니다. 이를 통해 데이터베이스를 깔끔하게 유지하고 불필요한 데이터를 관리합니다.

## 2. 소스 코드 분석

### 2.1 서버 구성

1. 서버 실행
   서버는 `app.js` 파일을 통해 실행되며, `express`, `morgan`, `nunjucks`, `sequelize` 등의 라이브러리를 사용합니다. `morgan`은 주석 처리되어 로그 기록 기능이 비활성화되어 있습니다.

2. 폴더 구조

- `config`: 데이터베이스 연결 정보를 포함한 설정 파일이 위치합니다. 예를 들어, 데이터베이스 접속 정보와 비밀번호 등이 저장되어 있습니다.
- `models`: Sequelize ORM을 사용하여 정의된 데이터베이스 모델 파일이 저장됩니다. 이 폴더에는 데이터베이스 테이블 구조를 정의하는 파일이 포함됩니다.
- `node_modules`: 프로젝트에서 사용하는 외부 라이브러리 및 모듈이 저장되는 공간입니다.
- `public`: 프론트엔드에서 사용하는 정적 파일(css, js, 이미지 등)이 저장됩니다.
- `routes`: Express 라우터 파일이 저장되는 폴더로, 각종 API 경로 및 요청 처리를 담당합니다.
- `views`: Nunjucks를 사용한 HTML 템플릿 파일이 저장됩니다.

### 2.2 데이터 파일 입력

업로드된 파일을 줄 단위로 분리하고, 각 줄을 공백, 탭, 콤마, 슬래시를 기준으로 분리하여 2차원 배열로 변환합니다. 이 배열을 서버로 전송하여 데이터베이스에 저장합니다. 서버는 파일 이름을 확인하여 동일한 이름의 테이블이 이미 있는지 검사한 후, 새 테이블을 생성하여 데이터를 저장합니다. 형식이 올바르지 않은 데이터는 무시되고, 유효한 데이터만 데이터베이스에 저장됩니다.

### 2.3 차트 출력

사용자가 데이터 리스트에서 특정 데이터를 선택하면, 서버에 `Core`와 `Task` 정보를 요청합니다. 서버는 데이터베이스에서 해당 정보를 조회하여 반환합니다. 반환된 데이터를 바탕으로 프론트엔드에서 `Chart.js` 라이브러리를 사용하여 차트를 생성합니다. 사용자는 차트 유형(예: 막대형, 선형)을 선택하여 데이터를 시각화할 수 있습니다.

# 3. 코드 개발

## 3.1 프로그램 개요

이 프로젝트는 사용자가 업로드한 데이터 파일을 처리하여 MongoDB에 저장하고, 저장된 데이터를 웹 인터페이스를 통해 시각화하는 웹 애플리케이션입니다. 이를 통해 CPU 코어의 성능과 각 작업(Task)의 결과를 한눈에 확인할 수 있습니다. Docker를 사용하여 데이터베이스와 서버를 효율적으로 관리합니다.

## 3.2 사용 방법 안내

1. Docker 환경설정 및 실행
   Docker와 Docker Compose를 사용하여 로컬 환경에 MongoDB와 Node.js 서버의 컨테이너를 설정합니다. `docker-compose.yml` 파일에 정의된 서비스를 통해 데이터베이스와 애플리케이션 서버가 함께 실행됩니다.
2. 프로젝트 시작
   터미널에서 `docker-compose up –build` 명령을 실행하여 모든 컨테이너를 시작합니다. 이 명령은 Docker Compose 파일에 정의된 설정에 따라 서버와 데이터베이스를 동시에 실행합니다.
   이후, 웹 브라우저를 통해 `http://localhost:3000`으로 접속하여 프로젝트의 기능을 테스트하고 사용할 수 있습니다.


**[**그림 **1]** 서버 실행 후 웹 브라우저 화면

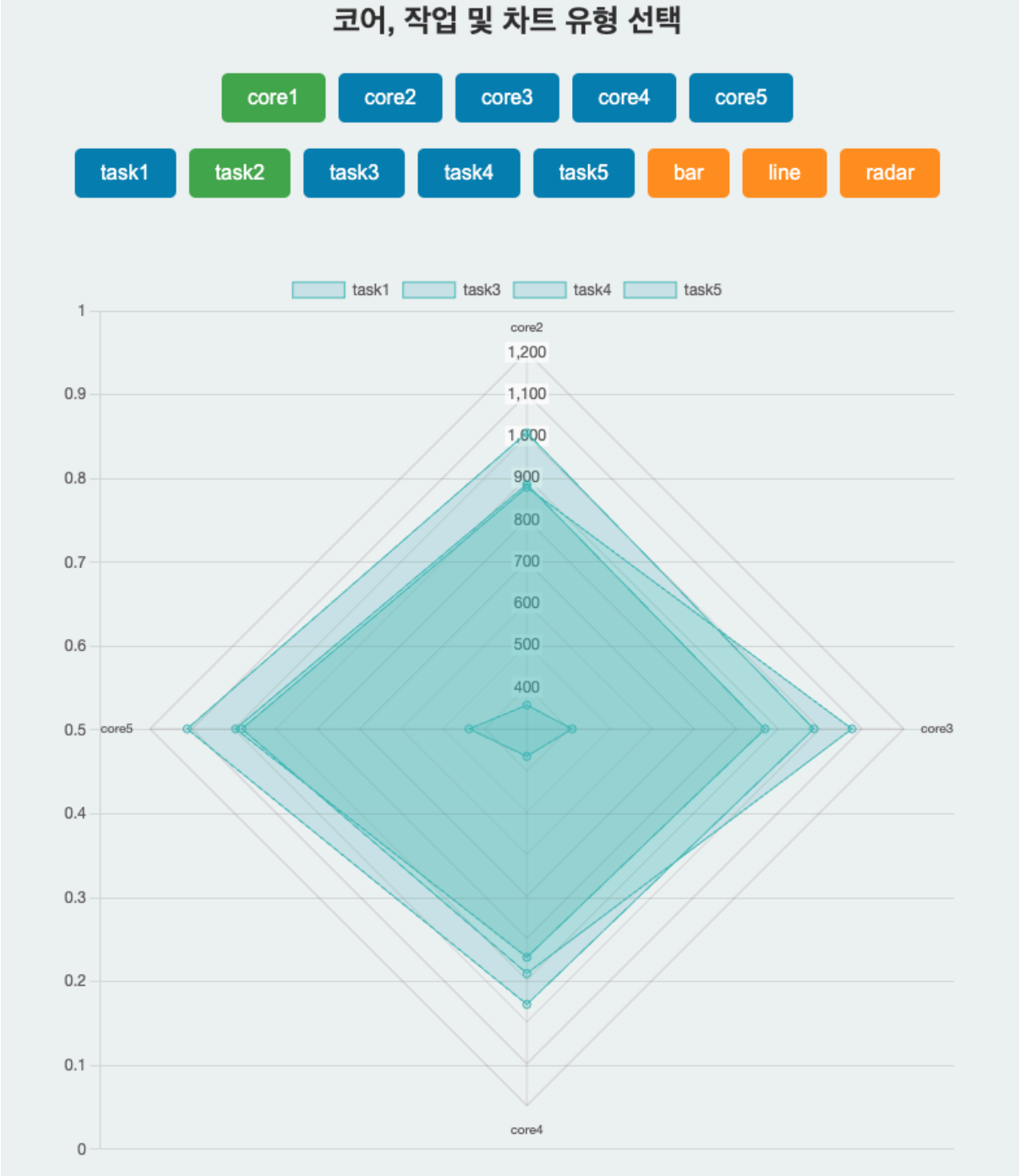
**3.3** 데이터 처리 및 입력

사용자는 웹 인터페이스를 통해 데이터 입력 화면에서 "파일 선택" 버튼을 클릭하여 업로드할 데이터 파일을 선택하고 제출합니다. Node.js 서버는 업로드된 파일을 처리하여 데이터를 MongoDB에 저장합니다. 파일 내의 데이터가 올바른 포맷으로 제공되지 않는 경우 오류 처리를 통해 건너뛰어집니다.

**3.4** 결과 출력 및 시각화

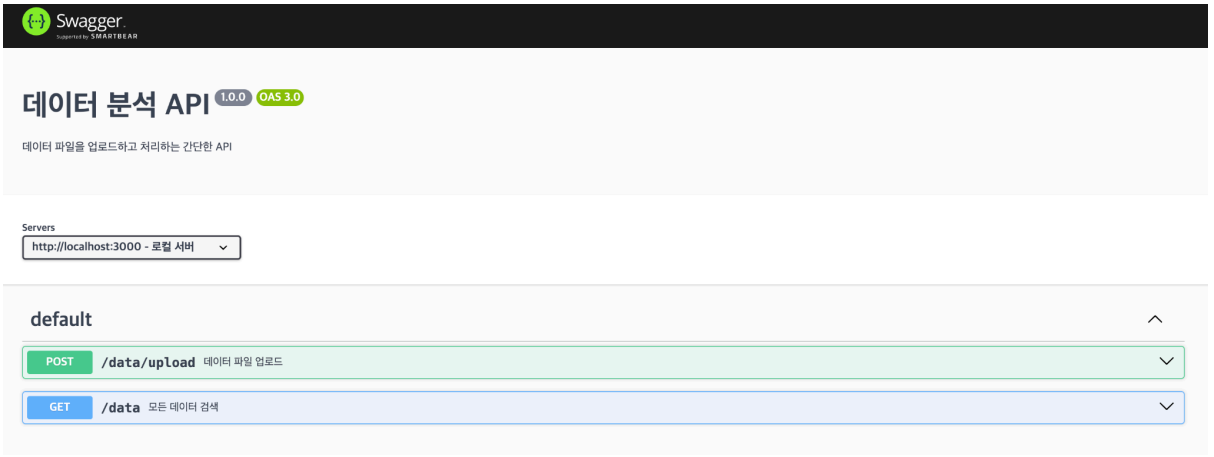
1. 차트 생성
   데이터가 정상적으로 저장된 후, 사용자는 Core와 Task를 선택하여 원하는 데이터를 차트로 확인할 수 있습니다. 차트 유형(바 차트, 라인 차트 등)을 선택하여 데이터를 시각화할 수 있습니다. 사용자가 데이터를 선택하면, 차트 유형 및 Core와 Task를 선택할 수 있는 버튼이 동적으로 생성되고 이를 통해 사용자는 더욱 유연하게 데이터를 조회하고 시각화할 수 있습니다. 이는 Chart.js 라이브러리를 활용하여 프론트엔드에서 데이터를 그래픽으로 변환합니다.


**[**그림 **2]** 데이터 결과 **: bar** 형태 차트




**[**그림 **3]** 데이터 결과 **: radar** 형태 차트

2. 스웨거를 통한 API 문서화
   스웨거를 사용하여 REST API에 대한 문서를 자동 생성하고, API 테스트 인터페이스를 제공합니다.




**[**그림 **4]** 스웨거를 통한 **API** 문서화

### **3.5** 소스 코드 분석

1. 서버 구성
    - Node.js 서버
        - Express 프레임워크를 사용하여 REST API를 구현됩니다. Docker 컨테이너 내에서 Node.js 환경이 구성되어 있습니다.
    - 데이터 처리
        - 업로드된 파일을 읽고 분석하여 MongoDB에 저장합니다.
        - Mongoose 라이브러리를 통해 데이터 스키마를 정의하고 데이터베이스 상호작용을 쉽게 처리합니다.
    - 차트 생성
        - Chart.js 라이브러리를 사용하여 데이터를 시각화합니다.
    - 사용자 인터페이스
        - EJS 템플릿 엔진을 사용하여 서버 사이드 렌더링을 통한 동적 페이지를 제공합니다.
        - 데이터와 상호작용하는 웹 인터페이스를 통해 사용자는 쉽게 데이터를 업로드하고 결과를 시각화할 수 있습니다.
    -
2. Docker 환경
    - Docker Compose:서비스 관리를 위해 `docker-compose.yml` 파일을 사용하여 MongoDB와 Node.js 애플리케이션을 정의하고 연결합니다.
    - 모든 서비스가 `app-network`라는 동일한 Docker 네트워크 내에서 통신할 수 있도록 설정합니다.
    - 데이터 지속성을 위해 MongoDB 데이터는 `mongo-data` 볼륨에 저장됩니다.

```yaml
version: "3.8"
services:
  app:
    container_name: node_app
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - .:/usr/src/app
    environment:
      - NODE_ENV=production
    depends_on:
      - mongo
    networks:
      - app-network

  mongo:
    container_name: mongo_db
    image: mongo:latest
    volumes:
      - mongo-data:/data/db
    ports:
      - "27017:27017"
    networks:
      - app-network

volumes:
  mongo-data:

networks:
  app-network:
    driver: bridge

```

**[그림 5] docker-compose.yml 파일**

https://github.com/fallkim/web-node