

웹 응용 기술

- Profiler 분석 -



과 목	웹응용기술
담당 교수	강영명 교수님
팀 장	20210853 여준

1. 프로그램 개요

1.1 개요

이 프로젝트는 웹 브라우저를 통해 데이터를 업로드하고, 대규모 데이터를 생성하여 분석 및 시각화하는 웹 애플리케이션이다. 사용자는 업로드한 파일 데이터를 분석하거나, 자동 생성된 대규모 데이터를 분석하여 데이터를 시각화할 수 있다. 주요 기능은 데이터의 최소값, 최대값, 평균 및 표준편차를 계산하고, 이를 그래프로 시각화하여 사용자에게 제공한다. 이를 통해 사용자는 데이터의 분포와 특성을 쉽게 이해할 수 있다.

1.1 주요 목적

- **파일 업로드:** Multer를 사용하여 파일을 서버에 업로드하고, 업로드된 파일의 경로를 가져온다. 파일을 읽어 데이터베이스에 저장하기 위한 준비를 한다.
- **데이터 처리:** 파일에서 읽은 데이터를 각 행(row)별로 분리하고, 각 행의 데이터를 숫자형 배열로 변환한다. 각 행의 MIN, MAX, AVG를 계산한다.
- **데이터베이스 저장:** 계산된 통계 데이터를 MySQL 데이터베이스에 저장한다. 저장된 데이터를 클라이언트로 전송하여 결과를 표시한다.
- **시각화:** 템플릿 엔진을 사용하여 결과를 웹 페이지에 시각화한다. 사용자는 각 Task와 Core에 대한 데이터를 다양한 그래프로 확인할 수 있다.

1.2 시스템 구성

- **프론트엔드:** HTML과 자바스크립트를 사용하여 사용자 인터페이스를 구성한다. 사용자는 파일 업로드 폼을 통해 데이터를 업로드하고, 버튼 클릭으로 대규모 데이터를 생성할 수 있다.
- **백엔드:** Express.js를 사용하여 서버를 구성하고, Multer를 사용하여 파일 업로드를 처리한다. MySQL 데이터베이스와 연동하여 업로드된 데이터 및 분석 결과를 저장하고, 필요한 통계 정보를 클라이언트에 제공한다.

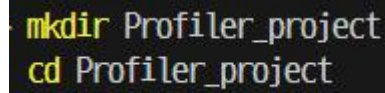
1.3 기술적 상세

- **Express.js:** Node.js 기반의 웹 애플리케이션 프레임워크로, HTTP 요청을 처리하고 서버를 구축하는 데 사용된다.
- **Multer:** 파일 업로드를 처리하는 미들웨어로, 업로드된 파일을 지정된 디렉토리에 저장한다.
- **MySQL:** 관계형 데이터베이스로, 분석된 통계 정보를 저장하고, 클라이언트 요청 시 데이터를 제공한다.
- **Chart.js:** 자바스크립트 기반의 차트 라이브러리로, 클라이언트에서 데이터를 시각화하는 데 사용된다.

2. 프로그램 수행 절차 분석

2.1 프로젝트 생성 및 초기화

2.1.1 디렉토리 생성 및 이동

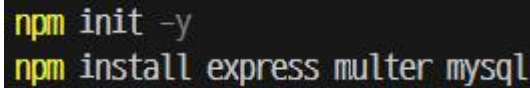


```
mkdir Profiler_project
cd Profiler_project
```

[그림 2.1.1] 디렉토리 생성 및 이동

‘Profiler_project’ 디렉토리를 생성하고 해당 디렉토리로 이동한다.

2.1.2 패키지 설치



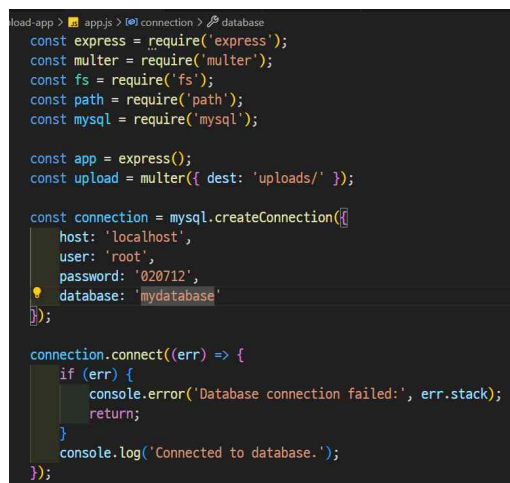
```
npm init -y
npm install express multer mysql
```

[그림 2.1.2] 패키지 설치

‘npm’을 사용하여 프로젝트를 초기화하고 필요한 패키지를 설치한다.

2.2 app.js

2.2.1 기본 설정 및 모듈 로드



```
load-app > app.js > connection > database
const express = require('express');
const multer = require('multer');
const fs = require('fs');
const path = require('path');
const mysql = require('mysql');

const app = express();
const upload = multer({ dest: 'uploads/' });

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '020712',
  database: 'mydatabase'
});

connection.connect((err) => {
  if (err) {
    console.error('Database connection failed:', err.stack);
    return;
  }
  console.log('Connected to database.');
```

[그림 2.2.1] 기본 설정

필요한 모듈을 로드하고, Express 애플리케이션을 설정하며, MySQL 데이터베이스에 연결한다. multer를 사용하여 파일 업로드 기능을 처리한다. express는 웹 서버를 구성하고, multer는 파일 업로드를, fs와 path는 파일 시스템 접근을, mysql은 MySQL 데이터베이스 연결을 관리한다. connection.connect는 데이터베이스 연결을 설정하고, 실패 시 오류를 출력한다.

2.2.2 표준편차 계산 함수

```
function calculateStandardDeviation(numbers) {  
  const mean = numbers.reduce((a, b) => a + b, 0) / numbers.length;  
  const variance = numbers.reduce((a, b) => a + Math.pow(b - mean, 2), 0) / numbers.length;  
  return Math.sqrt(variance);  
}
```

[그림 2.2.2] 표준편차

이 함수는 숫자 배열의 표준편차를 계산한다. 평균을 구한 후, 각 값이 평균에서 얼마나 떨어져 있는지 계산하여 분산을 구하고, 그 분산의 제곱근을 반환한다. 이를 통해 데이터의 분포를 이해할 수 있다.

2.2.3 정적 파일 제공 및 기본 라우트 설정

```
app.use(express.static(path.join(__dirname)));  
  
app.get('/', (req, res) => {  
  res.sendFile(path.join(__dirname, 'index.html'));  
});
```

[그림 2.2.3] 파일 제공 및 라우트 설정

이 부분에서는 정적 파일을 제공하고, 기본 라우트를 설정하여 index.html 파일을 제공한다. express.static을 사용하여 정적 파일을 제공하며, 기본 경로로 접속 시 index.html 파일을 반환한다.

2.2.4 파일 업로드 및 데이터 처리

```
app.post('/upload', upload.single('inputFile'), (req, res) => {
  const filePath = path.join(__dirname, req.file.path);
  fs.readFile(filePath, 'utf8', (err, data) => {
    if (err) {
      console.error('File read error:', err);
      return res.status(500).send('File read error');
    }

    try {
      const lines = data.split('\n');
      const numbers = [];

      lines.forEach(line => {
        const values = line.split(/\s+/).filter(v => v && !isNaN(v)).map(Number);
        numbers.push(...values);
      });

      if (numbers.length === 0) {
        return res.status(400).send('No numeric data found in the file');
      }

      const min = Math.min(...numbers);
      const max = Math.max(...numbers);
      const avg = numbers.reduce((a, b) => a + b, 0) / numbers.length;
      const stddev = calculateStandardDeviation(numbers);

      connection.query('INSERT INTO stats (min, max, avg) VALUES (?, ?, ?)', [min, max, avg], (error, results) => {
        if (error) {
          console.error('Database error:', error);
          return res.status(500).send('Database error');
        }
        res.json({ min, max, avg, stddev });
      });
    } catch (parseError) {
      console.error('Data processing error:', parseError);
      res.status(500).send('Data processing error');
    }
  });
});
```

[그림 2.2.4] 파일 업로드 및 데이터 처리

업로드된 파일을 읽고, 각 줄의 숫자 데이터를 파싱하여 최소값, 최대값, 평균 및 표준편차를 계산한다. 계산된 결과는 MySQL 데이터베이스에 저장하고 클라이언트에 반환한다. `upload.single('inputFile')`은 단일 파일 업로드를 처리하고, `fs.readFile`로 파일을 읽어 데이터를 파싱한다. `Math.min`, `Math.max` 및 `reduce`를 사용하여 통계 값을 계산하고, MySQL 쿼리로 데이터베이스에 저장한다.

2.2.5 대규모 데이터 생성 및 분석

```
app.get('/generate-large-data', async (req, res) => {
  const batchSize = 100000;
  const totalSize = 1000000;
  let largeData = [];
  let min = Infinity;
  let max = -Infinity;
  let sum = 0;

  for (let i = 0; i < totalSize; i += batchSize) {
    let batch = [];
    for (let j = 0; j < batchSize; j++) {
      const value = Math.floor(Math.random() * 10000);
      batch.push(value);
      if (value < min) min = value;
      if (value > max) max = value;
      sum += value;
    }
    largeData = largeData.concat(batch);
    await new Promise(resolve => setImmediate(resolve));
  }

  const avg = sum / largeData.length;
  const stddev = calculateStandardDeviation(largeData);

  res.json({ min, max, avg, stddev });
});
```

[그림 2.2.5] 대규모 데이터 생성 및 분석

이 엔드포인트는 대규모 데이터를 비동기적으로 생성한다. 데이터를 일정 크기 (batchSize)로 나누어 생성하여 메모리 사용을 최적화하고, 생성된 데이터의 최소값, 최대값, 평균 및 표준편차를 계산하여 클라이언트에 반환한다. setImmediate를 사용하여 이벤트 루프를 통해 데이터를 비동기적으로 처리하여 메모리 문제를 방지한다.

2.2.6 서버 시작

```
app.listen(8080, () => {  
  console.log('Server started on http://localhost:8080');  
});
```

[그림 2.2.6] 서버 시작

이 부분은 Express 애플리케이션을 8080 포트에서 실행하도록 설정한다. app.listen 메서드를 사용하여 서버를 시작하며, 서버가 성공적으로 시작되면 콘솔에 "Server started on <http://localhost:8080>"라는 메시지를 출력한다. 이를 통해 사용자는 서버가 정상적으로 실행되고 있는지 확인할 수 있다.

2.3 index.html

2.3.1 파일 업로드 양식 및 결과 캔버스

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>파일 업로드 및 시각화</title>  
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>  
  </head>  
  <body>  
    <h1>파일 업로드</h1>  
    <form id="upload-form" action="http://localhost:8080/upload" method="post" enctype="multipart/form-data">  
      <input type="file" name="inputFile" />  
      <button type="submit">업로드</button>  
    </form>  
  
    <h2>파일 업로드 결과</h2>  
    <div>  
      <canvas id="uploadResultChart"></canvas>  
    </div>
```

[그림 2.3.1] 파일 업로드 양식

이 부분은 파일 업로드 폼과 업로드된 데이터의 결과를 표시할 캔버스를 정의한다. 사용자가 파일을 선택하고 업로드 버튼을 클릭하면, 폼 데이터가 /upload 엔드포인트로 전송되고, 업로드된 파일의 데이터가 서버에서 처리되어 클라이언트에 반환된다. 결과는 uploadResultChart 캔버스에 표시된다.

2.3.2 대규모 데이터 생성 버튼 및 결과 캔버스

```
<h1>대규모 데이터 생성</h1>
<button id="generate-data-btn">대규모 데이터 생성</button>

<h2>대규모 데이터 결과</h2>
<div>
  <canvas id="largeDataResultChart"></canvas>
</div>
```

[그림 2.3.2] 대규모 데이터 버튼

대규모 데이터를 생성하는 버튼과 그 결과를 표시할 캔버스를 정의한다. 사용자가 "대규모 데이터 생성" 버튼을 클릭하면, 서버에 /generate-large-data 요청이 전송되고, 생성된 데이터의 통계 정보가 클라이언트에 반환된다. 결과는 largeDataResultChart 캔버스에 표시된다.

2.3.3 자바스크립트 코드

```
<script>
  document.getElementById('upload-form').addEventListener('submit', function(event) {
    event.preventDefault();
    const formData = new FormData(this);

    fetch('http://localhost:8080/upload', {
      method: 'POST',
      body: formData
    })
    .then(response => response.json())
    .then(data => {
      drawChart('uploadResultChart', data);
    })
    .catch(error => console.error('Error:', error));
  });

  document.getElementById('generate-data-btn').addEventListener('click', function() {
    fetch('http://localhost:8080/generate-large-data')
    .then(response => response.json())
    .then(data => {
      drawChart('largeDataResultChart', data);
    })
    .catch(error => console.error('Error:', error));
  });

  function drawChart(chartId, data) {
    const ctx = document.getElementById(chartId).getContext('2d');
    new Chart(ctx, {
      type: 'bar',
      data: {
        labels: ['최소값', '최대값', '평균', '표준편차'],
        datasets: [{
          label: '통계',
          data: [data.min, data.max, data.avg, data.stddev],
          backgroundColor: ['red', 'blue', 'green', 'purple']
        }]
      }
    });
  }
</script>
</body>
</html>
```

[그림 2.3.3] 자바스크립트 코드

- upload-form의 submit 이벤트를 처리하여 폼이 제출되지 않도록 막고, fetch를 사용하여 서버에 파일 업로드 요청을 보낸다. 서버로부터 받은 데이터를 drawChart 함수를 사용하여 그래프로 그린다.
- generate-data-btn의 click 이벤트를 처리하여 서버에 대규모 데이터 생성을 요청하고, 받은 데이터를 drawChart 함수로 시각화한다.
- drawChart 함수는 Chart.js 라이브러리를 사용하여 주어진 chartId 캔버스에 통계 데이터를 바 차트로 그린다.