모바일 프로그래밍 팀 과제 최종보고서

과목	웹응용기술
담당 교수님	강영명 교수님
학과	컴퓨터공학과
학번	20200882
이름	황인태

< 목차 >

1. 프로그램 개요

My Profiler는 Node.js를 활용하여 파일 업로드, 데이터 가공, 결과 시각화를 수행하는 웹 기반 애플리케이션입니다. 이 프로그램은 사용자가 업로드한 텍스트 파일 데이터를 서버에서 처리하여 데이터의 최소값(MIN), 최대값(MAX), 평균값(AVG), 표준편차(Standard Deviation)을 계산하고, 이를 웹 브라우저를 통해 시각적으로 표현합니다.

1.1 주요 기능

1. **파일 업로드**: 사용자가 웹 인터페이스를 통해 텍스트 파일(inputFile.txt)을 업로드할 수 있습니다.
2. **데이터 가공**: 서버는 업로드된 파일의 데이터를 읽어 들여 각 데이터 항목에 대해 최소값, 최대값, 평균값, 표준편차를 계산합니다.
3. **결과 반환**: 계산된 결과는 JSON 형식으로 클라이언트(웹 브라우저)로 반환됩니다.
4. **결과 시각화**: 클라이언트는 반환된 데이터를 기반으로 그래프(히스토그램, 파이차트 등)를 통해 시각적으로 표현할 수 있습니다.

1.2 사용 기술

- **Node.js**: 서버 사이드 자바스크립트 런타임
- **Express**: 웹 서버 프레임워크로 사용
- **Multer**: 파일 업로드를 처리하기 위해 사용
- **Mongoose**: MongoDB 객체 데이터 모델링(ODM) 라이브러리
- **Chart.js**: 클라이언트 측 데이터 시각화를 위한 라이브러리
- **EJS**: 템플릿 엔진을 사용하여 동적 HTML 생성

1.3 프로그램 흐름

1. 웹 브라우저에서 사용자가 파일을 선택하고 업로드 버튼을 클릭합니다.
2. 서버는 업로드된 파일을 수신하고, 'uploads/' 디렉터리에 저장합니다.
3. 서버는 저장된 파일을 읽어 데이터를 가공하여 최소값, 최대값, 평균값, 표준편차를 계산합니다.
4. 서버는 가공된 데이터를 JSON 형식으로 클라이언트에 반환합니다.
5. 웹 브라우저는 반환된 데이터를 기반으로 그래프를 생성하여 시각적으로 표현합니다.

1.4 프로그램의 목적

My Profiler는 데이터를 간편하게 업로드하고 분석하며, 시각화할 수 있는 도구를 제공하여 사용자들이 데이터 분석을 더 직관적으로 이해할 수 있도록 돕습니다. 데이터 처리 및 시각화 과정을 자동화하여 데이터 분석의 효율성을 높이는 것을 목표로 합니다.

2. 프로그램 수행 절차 분석

2.1 프로젝트 설정 및 환경 구성

2.1.1 프로젝트 디렉터리 구조

```
my-profiler /
├──── config/
│     └──── database.js
├──── models/
│     └──── dataModel.js
├──── public/
│     ├──── index.html
│     └──── styles.css
├──── routes/
│     └──── index.js
├──── utils/
│     └──── dataProcessor.js
├──── views/
│     ├──── index.ejs
│     └──── result.ejs
├──── uploads/
├──── app.js
├──── package.json
└──── package-lock.json
```

2.1.2 환경 변수 설정

config/database.js 파일에서 데이터베이스 연결 정보를 설정합니다. 여기에는 개발 환경에 맞는 MongoDB 연결 설정을 포함합니다.

```
const mongoose = require('mongoose');
const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/my_profiler', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error(err.message);
    process.exit(1);
  }
};
module.exports = connectDB;
```

2.2 데이터베이스 설정

2.2.1 데이터베이스 연결

config/database.js 파일에서 MongoDB 데이터베이스를 설정하고 연결합니다. 데이터베이스는 MongoDB를 사용합니다.

2.2.2 데이터 모델 정의

models/dataModel.js 파일에서 데이터 모델을 정의합니다.

```
const mongoose = require('mongoose');
const DataSchema =new mongoose.Schema({
  value: {
    type: Number,
    required: true
  }
});
const Data = mongoose.model('Data', DataSchema);
module.exports = Data;
```

2.3 프로젝트 설치 및 실행

2.3.1 의존성 설치

프로젝트 루트 디렉토리에서 npm install 명령어를 실행하여 필요한 npm 패키지를 설치합니다.

```
npm install
```

2.3.2 애플리케이션 실행

설치가 완료되면, npm start 명령어로 애플리케이션을 시작합니다.

```
npm start
```

서버가 시작되면, 브라우저에서 http://localhost:3000에 접속합니다.

2.4 데이터 처리

2.4.1 데이터 업로드

사사용자는 웹 인터페이스를 통해 데이터 파일(inputFile.txt)을 업로드합니다. 파일 업로드는 Multer 미들웨어를 사용하여 처리됩니다.

2.4.2 데이터 분석 및 처리

업로드된 파일은 서버에서 처리되며, 주요 처리 작업은 다음과 같습니다:

- 파일을 읽고 데이터 파싱
- 각 데이터 항목의 최소값, 최대값, 평균값 계산

해당 로직은 routes/index.js에서 정의되어 있습니다.

```
const express = require('express');
const router = express.Router();
const multer = require('multer');
const fs = require('fs');
const path = require('path');
const { calculateStats } = require('../utils/dataProcessor');
const upload = multer({ dest: 'uploads/' });
router.post('/upload', upload.single('file'), async (req, res) => {
  const filePath = path.join(__dirname, '../uploads/', req.file.filename);
  const data = fs.readFileSync(filePath, 'utf8');
  const parsedData = data.split('\n').map(Number);
  const results = calculateStats(parsedData);
  res.render('result', { data: results });
});
module.exports = router;
```

2.5 결과 시각화

2.5.1 데이터 시각화

처리된 데이터는 클라이언트 측에서 다양한 그래프로 시각화됩니다. 그래프 라이브러리는 Chart.js를 사용하여 구현됩니다.

2.5.2 결과 출력

사용자는 웹 인터페이스에서 각 데이터의 최소값, 최대값, 평균값, 표준편차를 다양한 형태의 그래프로 확인할 수 있습니다.

2.6 예외 처리

2.6.1 입력 데이터 오류

입력된 데이터가 유효하지 않은 경우, 서버에서 오류 메시지를 반환하고 데이터베이스에 저장되지 않습니다.

```
const isValidData = (data) => {
  return data.every(item =>!isNaN(item));
};
if (!isValidData(parsedData)) {
  res.status(400).send('Invalid data format');
  return;
}
```

2.7 데이터 관리

2.7.1 데이터 삭제

사용자는 웹 인터페이스에서 데이터 삭제 버튼을 클릭하여 데이터베이스에 저장된 데이터를 삭제할 수 있습니다.

```
router.post('/delete', async (req, res) => {
    await DataModel.deleteMany({});
    res.send('All data deleted');
});
```

3. 소스 코드 분석

3.1 주요 파일 및 디렉터리 구조

```
my-profiler /
├──── config/
│     └──── database.js
├──── models/
│     └──── dataModel.js
├──── public/
│     ├──── index.html
│     └──── styles.css
├──── routes/
│     └──── index.js
├──── utils/
│     └──── dataProcessor.js
├──── views/
│     ├──── index.ejs
│     └──── result.ejs
├──── uploads/
├──── app.js
├──── package.json
└──── package-lock.json
```

3.2 주요 소스 코드 분석

3.2.1 app.js

애플리케이션의 진입점으로서 서버를 설정하고, 미들웨어와 라우트를 연결합니다.

```
const express = require('express');
const connectDB = require('./config/database');
const indexRouter = require('./routes/index');
const app = express();
// Connect to database
connectDB();
// Set up middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static('public'));
// Set up view engine
app.set('view engine', 'ejs');
app.set('views', './views');
// Set up routes
app.use('/', indexRouter);
// Start server
const PORT = process.env.PORT ||3000;
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

3.2.2 config/database.js

MongoDB 데이터베이스 연결 설정 파일입니다.

```
const mongoose = require('mongoose');
const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/my_profiler', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error(err.message);
    process.exit(1);
  }
};
module.exports = connectDB;
```

3.2.3 models/dataModel.js

데이터 모델을 정의하는 파일입니다.

```
const mongoose = require('mongoose');
const DataSchema =new mongoose.Schema({
  value: {
    type: Number,
    required: true
  }
});
const Data = mongoose.model('Data', DataSchema);
module.exports = Data;
```

3.2.4 routes/index.js

라우터 파일로, 파일 업로드와 데이터 처리를 담당합니다.

```
const express = require('express');
const router = express.Router();
const multer = require('multer');
const fs = require('fs');
const path = require('path');
const { calculateStats } = require('../utils/dataProcessor');
const upload = multer({ dest: 'uploads/' });
router.post('/upload', upload.single('file'), async (req, res) => {
  const filePath = path.join(__dirname, '../uploads/', req.file.filename);
  const data = fs.readFileSync(filePath, 'utf8');
  const parsedData = data.split('\n').map(Number);
  const results = calculateStats(parsedData);
  res.render('result', { data: results });
});
module.exports = router;
```

3.2.5 utils/dataProcessor.js

데이터를 처리하고 통계를 계산하는 유틸리티 파일입니다.

```
const calculateStats = (data) => {
  const min = Math.min(...data);
  const max = Math.max(...data);
  const avg = data.reduce((a, b) => a + b, 0) / data.length;
  const stdDev = Math.sqrt(data.map(x => Math.pow(x - avg, 2)).reduce((a, b)
=> a + b) / data.length);
  return {
    min,
    max,
    avg,
    standardDeviation: stdDev
  };
};
module.exports = { calculateStats };
```

3.3 데이터 처리 로직

데이터 처리 로직은 utils/dataProcessor.js 파일에서 수행되며, 주어진 데이터 배열에 대해 최소값, 최대값, 평균값, 표준편차를 계산합니다.

3.4 클라이언트 측 코드

클라이언트 측 코드는 public 디렉터리의 정적 파일들과 views 디렉터리의 EJS 템플릿 파일들로 구성됩니다.

3.4.1 index.ejs

파일 업로드 폼을 제공하는 메인 페이지입니다.

```
<!DOCTYPE html >
<html lang ="en">
<head >
  <meta charset ="UTF-8">
  <meta name ="viewport"content ="width=device-width, initial-scale=1.0">
  <title >File Upload </title >
  <link rel ="stylesheet" href ="/styles.css">
</head >
<body >
  <div class ="container">
    <h1 >Upload your data file </h1 >
    <form action ="/upload" method ="POST" enctype
="multipart/form-data">
      <input type ="file"name ="file" required >
      <button type ="submit">Upload </button >
    </form >
  </div >
</body >
</html >
```

3.4.2 result.ejs

결과를 시각화하는 페이지입니다.

```
<!DOCTYPE html >
<html lang ="en">
<head >
  <meta charset ="UTF-8">
  <meta name ="viewport"content ="width=device-width, initial-scale=1.0">
  <title >Results </title >
  <link rel ="stylesheet" href ="/styles.css">
  <script src ="https://cdn.jsdelivr.net/npm/chart.js"></script >
</head >
<body >
  <div class ="container">
    <h1 >Data Processing Results </h1 >
    <p >Minimum: <%= data.min %></p >
    <p >Maximum: <%= data.max %></p >
    <p >Average: <%= data.avg %></p >
    <p >Standard Deviation: <%= data.standardDeviation %></p >
    <canvas id ="myChart"></canvas >
    <a href ="/">Upload Another File </a >
  </div >
  <script >
    const ctx =document.getElementById('myChart').getContext('2d');
    const chartData = {
      labels: ['Min', 'Max', 'Avg', 'Standard Deviation'],
      datasets: [{
        label: 'Data Stats',
        data: [<%= data.min %>, <%= data.max %>, <%= data.avg %>, <%=
data.standardDeviation %>],
        backgroundColor: [
          'rgba(255, 99, 132, 0.2)',
          'rgba(54, 162, 235, 0.2)',
          'rgba(255, 206, 86, 0.2)',
          'rgba(75, 192, 192, 0.2)'
        ],
        borderColor: [
          'rgba(255, 99, 132, 1)',
          'rgba(54, 162, 235, 1)',
          'rgba(255, 206, 86, 1)',
          'rgba(75, 192, 192, 1)'
        ],
        borderWidth: 1
      }]
    };
    const config = {
      type: 'bar',
      data: chartData,
      options: {
        scales: {
```

```
            y: {
              beginAtZero: true
            }
          }
        }
    };
    const myChart =new Chart(ctx, config);
  </script >
</body >
</html >
```