



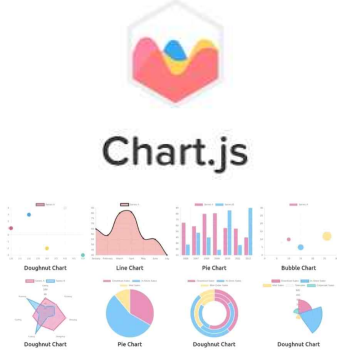
2023학년도 1학기 자바웹애플리케이션

Node.js를 활용한 Profiler 구현



종강언제오조	
20211026	조하나(팀장)
20210052	김미영
20201069	송유림
20210848	시소연
20210870	정보라

1. 전체 시스템 동작 파악(프로그램 수행 절차)

프로파일링	데이터 저장	시각화
		
수집 데이터 업로드	입력 포맷(txt, JSON)	서버 관리
프로파일러 On/Off	DB(SQL) 저장	원본 데이터 관리

1.1. DB 연결

1.1.1. 사전 준비

- nodegraph/app.js에서 202번째 줄의 인풋 텍스트 파일 저장 경로를 본인 환경에 맞게 변경
- MySQL 사용자 정보 설정 변경 / database 안의 테이블 모두 삭제

1.1.2. 실행 방법

- 터미널에서 nodegraph 경로로 들어가 node app 실행
- http://localhost:3000/upload에 접속해 inputfile.txt 파일을 업로드 (uploads 폴더에 저장해두었음)
- 이동한 화면에서 각 core1 ~ core5, task1 ~ task5 버튼을 누르며 그래프 결과 확인

1.1.3. 데이터 배열 만들기

- 3개의 테이블에서 core1의 task1의 평균 구하기
- 쿼리에서 UNION 이용해서 10개 테이블에서의 core1의 값들을 가져왔음
- rowValues[1] = core1의 task1의 값 이용해 평균 구함

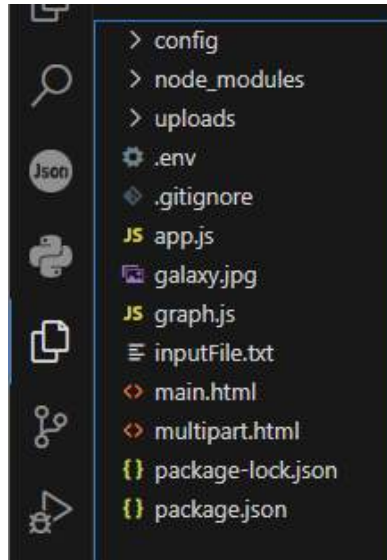
1.1.4. Express 라이브러리 설치 후 서버 구동하기

- 로컬파일 시스템에서 직접 HTML 파일을 열어 실행하면 오류가 발생하므로 Express 라이브러리 사용해 서버 생성
- 라우터를 사용해 app.js에서의 데이터를 graph.js로 전송하기
- 데이터가 json 형식으로 잘 전송되고 있는 화면

1.1.5. 입력파일 처리 및 MySQL DB 데이터 관리

- 업로드 된 파일을 읽어 가공한 후 DB 테이블 생성 및 데이터 삽입
- 가공한 파일 속 데이터를 MySQL 쿼리를 사용해 DB에 저장
- 전체 데이터가 담길 user 테이블과 core1~core5/task1~task5 데이터 관리를 위한 newtable1~newtable10, 총 11개의 테이블 생성

2. 프로젝트 파일 분석



[프로젝트 파일]

파일명	설명
app.js	입력파일을 처리하고 MySQL DB에 저장 MySQL에서 추출한 데이터를 JSON 형식으로 전달
galaxy.jpg	웹페이지 배경 이미지
graph.js	라우터에서 가져온 데이터를 JSON 형식으로 웹페이지에 전달 chart.js를 이용해 그래프 출력 html과 직접 연결하기 때문에 html에서 js를 불러오는 cdn형식 사용
inputFile.txt	입력 데이터 파일
main.html	입력받은 데이터를 그래프로 출력하는 html 파일
multipart.html	메인 html 파일
package-lock.json	package.json 파일과 관련된 추가적인 정보 포함 패키지 의존성의 정확한 버전과 해당 의존성이 의존하는 다른 패키지들의 구체적인 버전 정보 포함
package.json	현재 프로젝트에 대한 정보와 사용 중인 패키지에 대한 정보를 담은 파일 동일한 버전을 설치하지 않으면 문제 생길 수 있음 노드 프로젝트 시작 전 package.json부터 만들고 시작(npm init)

3. 코드 분석

3.1. 화면 설계

3.1.1. multipart.html: 메인 페이지



[입력화면]

```
55 <body>
56 <div id="wrap">
57 <div id="title">
58 <div>
59 <p style="margin-top: 40px; font-size: 20px; font-weight: bold;">[중강연제오조]</p>
60 <h1 style="margin-bottom: 40px; font-size: 60px;">파일 업로드</h1>
61 </div>
62 <div>
63 <form style="margin-bottom: 40px;" id="form" action="/upload" method="post" enctype="multipart/form-data">
64 <input type="file" name="text1" style="font-size: 20px; font-weight: bold;" />
65 <button type="submit" style="font-size: 20px; width: 100px;">업로드</button>
66 </form>
67 </div>
68 </div>
69 </div>
```

[html-body 코드]

<div id="title">	프로젝트의 팀명과 페이지의 제목을 안내
<form>	파일 업로드 기능이 구현될 레이아웃
graph.js	chart.js 라이브러리를 사용해 특정 데이터를 기반으로 그래프 그리 고 조작

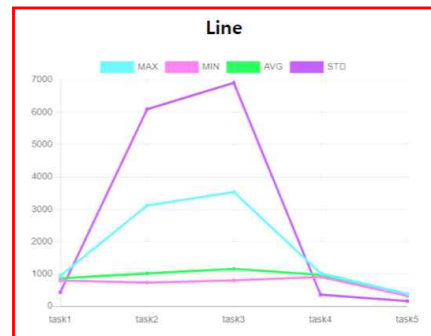
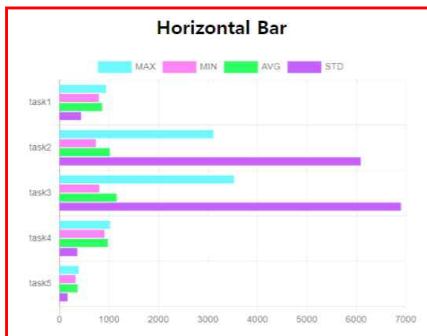
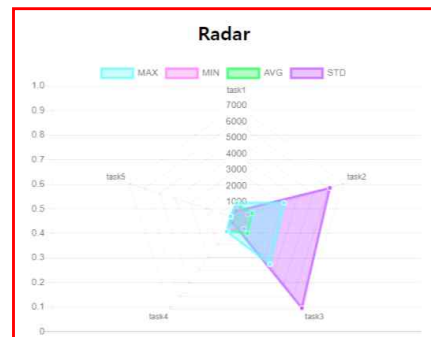
3.1.2. main.html: 그래프 구현

```
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
<script src="./graph.js"></script>
```

chart.js	라이브러리를 이용해 그래프 구현
graph.js	chart.js 라이브러리를 사용해 특정 데이터를 기반으로 그래프 그리고 조작



Core1의 Task별 수행능력



3.2. app.js : 서버 구동의 핵심이 되는 파일

```

5  const express = require('express');
6  const morgan = require('morgan');
7  const cookieParser = require('cookie-parser');
8  const session = require('express-session');
9  const dotenv = require('dotenv');
10 const path = require('path');
11 const app = express();
12 const router = express.Router(); // 라우터 객체 생성
13 app.set('port', process.env.PORT || 3000);
14 dotenv.config();

```

require('express')	Express 애플리케이션 생성 및 서버 구성
require('morgan')	HTTP 요청에 대한 로깅 제공하는 Morgan 미들웨어
require('cookie-parser')	요청된 쿠키를 쉽게 추출하고 다루기 위한 미들웨어인 cookieParser를 불러옴. 이를 통해 쿠키를 파싱하고 사용할 수 있음
require('express-session')	세션 관리를 위한 미들웨어인 Express-session 불러옴. 세션은 클라이언트와 서버 간에 상태를 유지하기 위해 사용
require('dotenv')	환경 변수를 파일에서 읽어오기 위한 dotenv 불러옴
require('path')	파일 및 디렉터리 경로를 다루기 위한 내장 모듈이 path를 불러옴
app=express()	Express 애플리케이션 생성. 미들웨어 추가하고 라우팅 설정 가능
router=express.Router()	Express 라우터 객체 생성. URL 경로에 따라 요청 처리하는 데 사용
app.set()	app.set('port', 포트번호)로 서버가 실행될 기본값 3000세팅

3.2.1. 익스프레스 서버 실행하기

```
PS C:\node\nodegraph> npm start
```

```
> npmtest@1.0.0 start
> nodemon app
```

```

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
서버가 http://localhost:3000/upload 에서 실행 중입니다.

```

```

{
  "scripts": {
    "start": "nodemon app"
  },
}

```

- npm start(package.json의 start 스크립트) 콘솔에서 실행
>>localhost:3000
- 요청이 전송되고 응답이 왔을 때 쿠키가 설정됨

3.2.2. express는 미들웨어로 구성

```

16 app.use('/', router);
17 app.use(morgan('dev'));
18 app.use(express.static(path.join(__dirname, '/'))); // 정적 파일 제공
19 app.use(express.json());
20 app.use(express.urlencoded({ extended: false }));
21 app.use(cookieParser(process.env.COOKIE_SECRET));
22 app.use(session({
23   resave: false,
24   saveUninitialized: false,
25   secret: process.env.COOKIE_SECRET,
26   cookie: {
27     httpOnly: true,
28     secure: false,
29   },
30   name: 'session-cookie',
31 }));
32

```

app.use('/', router)	'/'로 들어오는 모든 요청에 대해 router 미들웨어 사용
app.use(morgan('dev'))	개발 환경에서 HTTP 요청에 대한 로깅 제공하는 Morgan 미들웨어 사용
app.use(express.static(path.join(__dirname, '/')))	정적 파일 제공하기 위한 미들웨어 추가. express.static 함수를 사용해 정적 파일들이 위치한 디렉터리 지정
미들웨어	내부에서 알아서 next를 호출해서 다음 미들웨어로 넘어감
	파일을 발견했다면 다음 미들웨어는 실행되지 않음

body-parser: 요청의 본문을 해석해주는 미들웨어

app.use(express.json())	json 미들웨어는 요청 본문이 json인 경우 해석, urlencoded 미들웨어는 폼 요청 해석
app.use(express.urlencoded())	
app.use(cookieParser(process.env.COOKIE_SECRET))	쿠키를 파싱하기 위한 미들웨어 CookieParser 추가

express-session: 세션 관리용 미들웨어

app.use(session())	Express-session 미들웨어 사용하여 세션 쿠키에 대한 설정 (secret: 쿠키 암호화, cookie: 세션 쿠키 옵션)
resave	요청이 왔을 때 세션에 수정사항이 생기지 않아도 다시 저장할지 여부
saveUninitialized	세션에 저장할 내역이 없더라도 세션을 저장하지
httpOnly	true로 설정하여 클라이언트 측에서 쿠키 수정할 수 없음
secure	false로 설정하여 HTTPS가 아닌 환경에서도 쿠키 사용 가능

3.2.3. 쿠키

```
1 COOKIE_SECRET=cookiesecret
```

```
>>$npm i morgan cooki-parser express-session dotenv
```

3.2.4. 데이터베이스 생성

```
59 app.get('/upload', (req, res) => { //.../upload 로 이동 시 html 파일과 연결
60   res.sendFile(path.join(__dirname, 'multipart.html'));
61 });
62
63 app.post('/upload', upload.single('text1'), (req, res) => { // 인풋 텍스트
64
65   //db 스키마 이름: test , 기존 테이블 이름:user,
66   //분할된 테이블 10개의 이름: newtable1 ~ newtable10
67   const mysql = require('mysql');
68
69   // MySQL 데이터베이스 연결 설정
70   var connection = mysql.createConnection({
71     host      : 'localhost',
72     user      : 'root',
73     password  : '20211026',
74     database  : 'nodejs'
75   });
76
77   // db 연결
78   connection.connect((err) => {
79     if (err) throw err;
80     console.log('Connected to MySQL database');
81   });
```

```
83 const createTableQueries = [
84   //파일의 전체 데이터 값이 저장 될 user테이블
85   `CREATE TABLE user (
86     cno VARCHAR(45) NOT NULL,
87     task1 INT,
88     task2 INT,
89     task3 INT,
90     task4 INT,
91     task5 INT
92   );`,
93
94   `CREATE TABLE newtable1 (
95     cno VARCHAR(45) NOT NULL,
96     task1 INT,
97     task2 INT,
98     task3 INT,
99     task4 INT,
100    task5 INT
101  );`,
102
103   `CREATE TABLE newtable2 (
104     cno VARCHAR(45) NOT NULL,
105     task1 INT,
106     task2 INT,
107     task3 INT,
108     task4 INT,
109     task5 INT
110  );`,
```

db 스키마 이름: test
기존 테이블 이름: user
분할된 테이블 10개의 이름
:newtable1 ~ newtable10

app.get	/upload 경로로 GET 요청이 들어왔을 때 처리하는 함수. 클라이언
---------	--

(<code>('/upload',())</code>)	트에 <code>'multipart.html'</code> 파일을 보내주는 역할. <code>res.sendFile()</code> 함수를 사용하여 해당 html 파일의 경로를 지정하여 클라이언트에 전송
<code>app.post</code> (<code>('/upload',())</code>)	<code>/upload</code> 경로로 POST 요청이 들어왔을 때 처리하는 함수. <code>single</code> 메소드를 사용하여 <code>text1</code> 필드에 업로드된 단일 파일 처리. 파일 업로드 후에 실행되며, 파일의 저장 위치 설정과 관련된 로직 추가해야 함
<code>require('mysql')</code>	MySQL 데이터베이스와의 연결을 위해 <code>'mysql'</code> 모듈 불러옴.
<code>mysql.createConnection()</code>	MySQL 데이터베이스에 대한 연결 설정. 호스트, 사용자, 비밀번호, 데이터베이스 이름 등을 포함한 연결 설정 정보 객체 전달
<code>connection.connect(err)</code>	MySQL 데이터베이스에 연결하는 함수. 연결이 성공하면 <code>'Connected to MySQL database'</code> 라는 메시지 출력

3.2.5. inputFile.txt 저장할 폴더 생성

```

38 //inputfile 저장할 폴더 생성
39 try {
40   fs.readdirSync('uploads');
41 } catch (error) {
42   console.error('uploads 폴더가 없어 uploads 폴더를 생성합니다.');
```

<code>fs.readdirSync()</code>	현재 경로에 있는 <code>uploads</code> 폴더의 내용 읽어옴
<code>catch(error)</code>	동기적으로 실행되므로 폴더 없으면 예외 발생 예외 발생 시 <code>'uploads 폴더가 없어 upload 폴더를 생성합니다'</code> 출력
<code>fs.mkdirSync()</code>	현재 경로에 <code>uploads</code> 폴더 동기적으로 생성. 폴더가 없을 때 새로운 <code>uploads</code> 폴더 생성

3.2.6. Multer 미들웨어 설정

```

34 const multer = require('multer');
35 const fs = require('fs');
36 const database = require('mime-db');
```

<code>require('multer')</code>	파일 업로드 처리위해 Multer 미들웨어 불러옴
<code>require('fs')</code>	파일 시스템 다루기 위한 내장 모듈인 <code>fs</code> 불러옴. <code>fs</code> 모듈은 파일을 읽거나 쓰는 등의 파일 시스템 작업 수행 가능
<code>require('mime-db')</code>	파일 확장자와 Mime 유형 간의 매핑 정보를 제공하는 DB

```

46 const upload = multer({
47   storage: multer.diskStorage({
48     destination(req, file, done) {
49       done(null, 'uploads/');
50     },
51     filename(req, file, done) { //파일 이름 지정
52       const ext = path.extname(file.originalname);
53       done(null, path.basename(file.originalname, ext) + ext);
54     },
55   }),
56   limits: { fileSize: 5 * 1024 * 1024 }, // 파일 최대 크기 설정
57 });

```

storage: multer	저장할 공간에 대한 정보. 하드디스크에 업로드 파일을 저장한다는 것
.diskStorage	
destination()	파일이 저장될 경로 지정. 'uploads/'로 설정됨
filename()	저장할 파일명 지정(파일명+날짜+확장자 형식)
limits: {}	파일의 최대 크기 설정

```

63 app.post('/upload', upload.single('text1'), (req, res) => {

```

upload.single	하나의 파일 업로드할 때 사용, input.txt 파일 저장 위치 설정
---------------	---

3.2.7. 데이터베이스 테이블 생성

```

185 // 테이블 생성 함수
186 function createTables() {
187   for (let i = 0; i < createTableQueries.length; i++) {
188     connection.query(createTableQueries[i], (err, results) => {
189       if (err) {
190         console.error(`테이블 ${i + 1} 생성 실패:`, err);
191       } else {
192         console.log(`테이블 ${i + 1}이(가) 성공적으로 생성되었습니다.`);
193       }
194     });
195   }
196 }

```

function createTables()	테이블 생성 함수
connection.query()	connection 객체를 사용하여 SQL 쿼리 실행. query 메서드는 쿼리를 실행하고 결과를 받는 비동기 함수
(err, results)	쿼리 실행 결과를 처리하는 콜백 함수
if(err) else	쿼리 실행 중에 발생한 오류 여부 확인

```

198 // 테이블 생성 함수 실행
199 createTables();
200
201 //인풋 텍스트 파일 저장
202 connection.query("LOAD DATA LOCAL INFILE 'C:/node/nodegraph/uploads/
203
204 // MySQL 쿼리 실행 함수 (10개의 테이블로 데이터 전달)
205 function executeQueries() {
206     const queries = [
207         'INSERT INTO newtable1 SELECT * FROM user LIMIT 5',
208         'INSERT INTO newtable2 SELECT * FROM user LIMIT 5 OFFSET 5',
209         'INSERT INTO newtable3 SELECT * FROM user LIMIT 5 OFFSET 10',
210         'INSERT INTO newtable4 SELECT * FROM user LIMIT 5 OFFSET 15',
211         'INSERT INTO newtable5 SELECT * FROM user LIMIT 5 OFFSET 20',
212         'INSERT INTO newtable6 SELECT * FROM user LIMIT 5 OFFSET 25',
213         'INSERT INTO newtable7 SELECT * FROM user LIMIT 5 OFFSET 30',
214         'INSERT INTO newtable8 SELECT * FROM user LIMIT 5 OFFSET 35',
215         'INSERT INTO newtable9 SELECT * FROM user LIMIT 5 OFFSET 40',
216         'INSERT INTO newtable10 SELECT * FROM user LIMIT 5 OFFSET 45'
217     ];

```

- 테이블 생성 함수를 실행한 후 데이터를 MySQL 데이터베이스에 저장하고, 그 후에 10개의 테이블에 데이터를 전달하는 기능 수행

```

343 //task별 core데이터 추출 쿼리
344 const taskQueries = [
345     SELECT task1 FROM (
346         SELECT task1 FROM newtable1 UNION ALL
347         SELECT task1 FROM newtable2 UNION ALL
348         SELECT task1 FROM newtable3 UNION ALL
349         SELECT task1 FROM newtable4 UNION ALL
350         SELECT task1 FROM newtable5 UNION ALL
351         SELECT task1 FROM newtable6 UNION ALL
352         SELECT task1 FROM newtable7 UNION ALL
353         SELECT task1 FROM newtable8 UNION ALL
354         SELECT task1 FROM newtable9 UNION ALL
355         SELECT task1 FROM newtable10
356     ) AS task_data1
357 ],
358     SELECT task2 FROM (
359         SELECT task2 FROM newtable1 UNION ALL
360         SELECT task2 FROM newtable2 UNION ALL
361         SELECT task2 FROM newtable3 UNION ALL
362         SELECT task2 FROM newtable4 UNION ALL
363         SELECT task2 FROM newtable5 UNION ALL
364         SELECT task2 FROM newtable6 UNION ALL
365         SELECT task2 FROM newtable7 UNION ALL
366         SELECT task2 FROM newtable8 UNION ALL
367         SELECT task2 FROM newtable9 UNION ALL
368         SELECT task2 FROM newtable10
369     ) AS task_data2
370 ],
371 ];
372

```

```

269 //core별 task데이터 추출 쿼리
270 const coreQueries = [
271     SELECT * FROM (
272         SELECT * FROM newtable1 UNION ALL
273         SELECT * FROM newtable2 UNION ALL
274         SELECT * FROM newtable3 UNION ALL
275         SELECT * FROM newtable4 UNION ALL
276         SELECT * FROM newtable5 UNION ALL
277         SELECT * FROM newtable6 UNION ALL
278         SELECT * FROM newtable7 UNION ALL
279         SELECT * FROM newtable8 UNION ALL
280         SELECT * FROM newtable9 UNION ALL
281         SELECT * FROM newtable10
282     ) AS core_data WHERE cno = "core1"
283 ],
284     SELECT * FROM (
285         SELECT * FROM newtable1 UNION ALL
286         SELECT * FROM newtable2 UNION ALL
287         SELECT * FROM newtable3 UNION ALL
288         SELECT * FROM newtable4 UNION ALL
289         SELECT * FROM newtable5 UNION ALL
290         SELECT * FROM newtable6 UNION ALL
291         SELECT * FROM newtable7 UNION ALL
292         SELECT * FROM newtable8 UNION ALL
293         SELECT * FROM newtable9 UNION ALL
294         SELECT * FROM newtable10
295     ) AS core_data WHERE cno = "core2"
296 ],
297 ];
298
299

```



```

454 //coreResults와 taskResults를 담을 객체
455 var resData = {};
456
457 //모든 task쿼리가 실행되었으면 json형식으로 데이터 전달
458 if (taskResults.length === taskQueries.length) {
459     resData.core = coreResults;
460     resData.task = taskResults;
461
462     res.json({resData});
463 }

```

3.2.8. promise와 query

```

219 // Promise 로 비동기적 처리
220 // 테이블 생성 및 삽입 과정이 완전히 끝난 후 main.html이 실행되도록
221 // 쿼리 실행
222 const promises = queries.map((query, index) => {
223     return new Promise((resolve, reject) => {
224         connection.query(query, (err, result) => {
225             if (err) reject(err);
226             console.log(`Query ${index + 1} executed successfully`);
227             resolve();
228         });
229     });
230 });

```

- promise로 비동기적 처리
- 테이블 생성 및 삽입 과정이 완전히 끝난 후, main.html 실행
- 쿼리 실행

```

233 Promise.all(promises)
234 .then(() => {
235     console.log("실행 끝");
236     res.sendFile(path.join(__dirname, 'main.html'));
237 })
238 .catch((err) => {
239     console.error("쿼리 실행 중 에러 발생:", err);
240 });
241 }

```

- 모든 쿼리 실행이 완료된 후 main.html 실행

```

420 //쿼리 실행하며 coreResults와 taskResults에 데이터 저장
421 for (let i = 0; i < coreQueries.length; i++) {
422
423     //core쿼리 실행
424     connection.query(coreQueries[i], function(error, results, fields) {
425         if (error) throw error;
426
427         var coreData = [];
428         for (let j = 0; j < results.length; j++) {
429             var rowValues = Object.values(results[j]);
430             for (let k = 1; k < rowValues.length; k++) {
431                 coreData.push(rowValues[k]);
432             }
433         }
434         coreResults.push(coreData);

```

- 쿼리 실행하여 coreResults와 taskResult에 데이터 저장

3.2.9. 서버 실행

```
473 app.listen(app.get('port'), () => {
474   console.log('서버가 http://localhost:'
475     + app.get('port') + '/upload' + ' 에서 실행 중입니다. ');
476 });
```

app.listen()	app.listen('포트번호', 콜백)으로 몇 번 포트에서 서버를 실행할지 지정
	localhost:3000번에서 정상적으로 실행됨을 확인

3.3. graph.js

3.3.1. 라우터에서 가져온 데이터 웹상으로 띄우기

```
8 window.onload(sendAjax(1));
9
10 function sendAjax(btn) {
11   console.log("sendAjax()");
12
13   // 데이터 가져오기
14   fetch('/data').then((res) => res.json())
15     .then((res) => {
16
17     console.log("그래프 실행");
18     console.log("가져온 데이터 res:", res);
19
20     // 차트 제목 수정
21     var strTitle;
22     if(btn <= 5) { // core1~core5
23       strTitle = "Core" + btn + "의 Task별 수행능력";
24     } else { // task1~task5
25       btn = (btn%5)+1;
26       strTitle = "Task" + btn + "의 Core별 수행능력";
27     }
28     var title = document.getElementById('title');
29     title.textContent = strTitle;
```

window.onload(sendAjax)	JavaScript를 사용하여 웹 페이지의 초기화 시점인 window.onload 이벤트가 발생했을 때 sendAjax 함수 호출
fetch('/data')	서버에서 데이터를 가져오는 역할로 /data 경로로 GET 요청 보냄. 응답은 promise 객체로 반환
.then((res)=>res.json())	promise를 이용하여 응답을 JSON 형식으로 파싱
strTitle	task별, core별 수행능력 출력

3.3.2. 데이터 가공 수행 과정

```

50     var dataArray;
51     if(btn <= 5) { // core1~core5
52         dataArray = Object.values(res.resData.core);
53     } else { // task1~task5
54         dataArray = Object.values(res.resData.task);
55     }
56     var row = dataArray[btn-1];
57     var data = [];

```

1	core1~core5, task1~task5 가 차례대로 담긴 dataArray 만들기 EX) core2=[task1~task5, task1~task5, ...] 10개 테이블이 한 줄로 들어가있음
---	---

```

59     // task1~task5/core1~core5 나눠 저장하기
60     for(let i = 1; i <= 5; i++) {
61         col = [];
62         for(let j = i; j <= row.length; j+=5) {
63             col.push(row[j-1]);
64         }
65         data[i-1] = col;
66     }
67     console.log("data:", data);

```

2	눌린 버튼의 번호를 받아와서 거기에 있는 데이터 꺼내서 core/task 별로 나누기 EX) data = [[10개 테이블에서의 core2의 task1 값들], [10개 테이블에서의 core2의 task2 값들], ...];
---	--

```

87     function MAX(data) { // MAX
88         for(let i = 0; i < data.length; i++) { // task1~task5/core1~core5
89             maxList[i] = Math.max.apply(null, data[i]);
90         }
91     }
92     function MIN(data) { // MIN
93         for(let i = 0; i < data.length; i++) { // task1~task5/core1~core5
94             minList[i] = Math.min.apply(null, data[i]);
95         }
96     }
97     function AVG(data) { // AVG
98         for(let i = 0; i < data.length; i++) { // task1~task5/core1~core5
99             var temp = data[i];
100             for(let j = 0; j < data[0].length; j++) {
101                 avgList[i] += temp[j];
102             }
103             avgList[i] /= data[0].length;
104         }
105     }
106     function STD() { // STD
107         for(let i = 0; i < data.length; i++) { // task1~task5/core1~core5
108             var temp = data[i];
109             for(let j = 0; j < temp.length; j++) {
110                 dev = temp[j] - avgList[i];
111                 devTotal[i] += Math.pow(dev,2);
112             }
113         }
114         for (let i = 0; i < stdList.length; i++) {
115             stdList[i] = Math.sqrt(devTotal[i] / data.length);
116         }
117     }

```


3	꺼낸 데이터로 MAX, MIN, AVG, STD 구해서 개별 배열로 저장하기
---	--

3.3.3. 그래프 출력

```

119 // chart.js 그래프 띄우기
120 var label = ['task1', 'task2', 'task3', 'task4', 'task5'];
121
122 var ctx = document.getElementById('test1').getContext('2d');
123 var mychart = new Chart(ctx, {
124     type: 'bar',
125     data: {
126         labels: label,
127
128     }, {
129         label: 'MAX',
130         fill: false, // 채우기 없음
131         lineTension: 0, // 숫자가 높을 수록 둥글어짐
132         pointRadius: 2, // 각 지점에 포인트 주지 않
133         backgroundColor: 'rgba(108, 241, 255, 1)',
134         borderColor: 'rgba(108, 241, 255, 1)',
135         data: maxList
136     }, {
137         label: 'MIN',
138         display: false,
139         fill: false,
140         lineTension: 0,
141         pointRadius: 2,
142         backgroundColor: 'rgba(255, 127, 239, 1)',
143         borderColor: 'rgba(255, 127, 239, 1)',
144         data: minList
145     }, {
146         label: 'AVG',
147         display: false,
148         fill: false,
149         lineTension: 0,
150         pointRadius: 2,
151         backgroundColor: 'rgba(44, 255, 94, 1)',
152         borderColor: 'rgba(44, 255, 94, 1)',
153         data: avgList
154     }, {
155         label: 'STD',
156         display: false,
157         fill: false,
158         lineTension: 0,
159         pointRadius: 2,
160         backgroundColor: 'rgba(192, 95, 255, 1)',
161         borderColor: 'rgba(192, 95, 255, 1)',
162         data: stdList
163     }
164 });

```

label	chart.js 그래프 띄우기 task별로 data의 MAX,MIN,AVG,STD 생성
-------	---

4. 기여도 (동일)

- 조하나, 김미영, 송유림, 시소연, 정보라