

2025-02-01

~

2025-02-07

조회

(%)

(Byte)

2025. Team Project

Network Dashboard

네트워크 대시보드

status

S_CNT

LAST_LOGIN_TIME

2025-02-07 12:39:33

2025-02-06 15:56:51

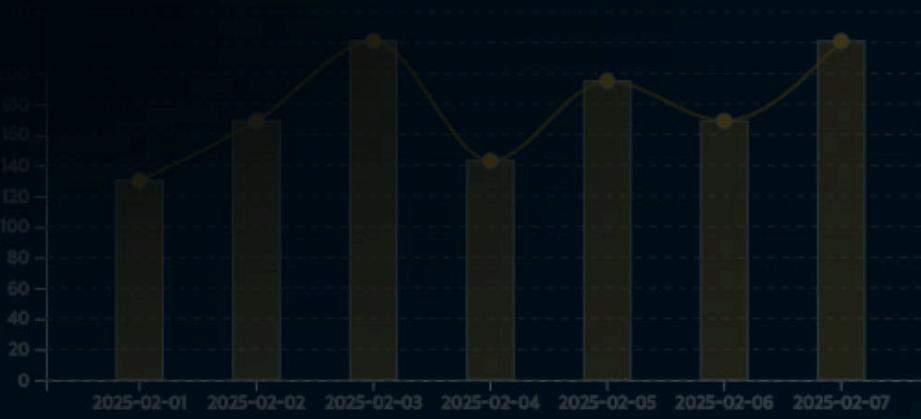
Daily Memory Swap Usage(%)



Daily CPU Usage(%)



Daily Website Visitor Count



Daily Service Error Count



Weekly Critical Error Log Status

서동현, 배주환, 서현지, 윤혜경

LOG_TIME	SERVICE	LOG_LEVEL	MESSAGE
20250202	ufw	CRITICAL	ufw: Denied access from 15.165.25.114

Index

1. Overview	프로젝트 개요	3p - 10p
2. Timeline	일정	11p - 12p
3. Stack	기술	13p - 16p
4. Wireframe	화면 설계	17p - 21p
5. UI design	UI 디자인	22p - 29p
6. Code Analysis	코드 분석	30p - 52p
6-1. Front end	프론트엔드 코드 분석	33p - 41p
6-2. Back end	백엔드 코드 분석	42p - 54p
7. Review	후기	55p - 59p

Overview

프로젝트 개요

1. Overview

소개

“

서버와 네트워크를 한 눈에,
쉽고 직관적인 **모니터링 웹 서비스**

”

1. Overview

개요

시스템 구축의 어려움 해결

최근 기업들은 자체 데이터센터를 운영하기보다 클라우드 서비스를 활용하여 웹 및 앱을 운영하는 추세입니다.

그러나 규모가 작은 기업이나 개인 서버 운영자들은 대기업처럼 고도화된 모니터링 시스템을 구축하기 어렵습니다.

사용자 친화적인 서버 모니터링 서비스 제공

이러한 문제를 해결하기 위해 **서버 운영 경험이 부족한 사용자와 소규모 기업을 대상으로,** 서버 상태와 네트워크 상황을 **직관적으로 모니터링하고 손쉽게 제어할 수 있는 웹서비스를 개발하고자 합니다.**



1. Overview

목적



여러 개의 서버 상태를 명령어를 입력하지 않고
한눈에 확인할 수 있는 서비스 제공



서버 및 네트워크 상태를 **쉽고 직관적으로**
모니터링 할 수 있는 사용자 친화적 인터페이스 제공



서버 운영 경험이 적은 사용자도
쉽고 편리하게 관리할 수 있도록 서비스 지원

1. Overview

목표

실시간 모니터링

하나의 웹사이트에서
여러 개의 서버 상태를 실시간으로
모니터링할 수 있도록 구현

간편한 제어 기능

간단한 제어 기능을 추가하여,
서버에 직접 접속하지 않고도
기본적인 관리가 가능하도록 지원



직관적 UI/UX

직관적인 UI/UX를 통해 사용자가
쉽게 서버의 자원 및 네트워크
상태를 확인할 수 있도록 개발

AI 기반 챗봇

사용자가 네트워크 및 서버 관련
궁금한 점을 해결할 수 있도록
AI 기반 챗봇 기능을 제공

1. Persona

페르소나



소규모 벤처기업 대표 - 나사장 (45세)

“새로 시작한 강원도 여행지 추천 서비스의 서버가 자꾸 불안정해.
매번 직접 서버에 들어가서 확인하는 게 너무 번거롭고 힘들어.
한눈에 서버 상태를 확인할 수 있는 모니터링 서비스가 있으면 좋겠어.”

- 안정적인 서버 운영을 위해 여러 서버의 상태를 한눈에 파악하고 싶음
- 직접 명령어를 입력하지 않고도 직관적인 모니터링이 가능해야 함

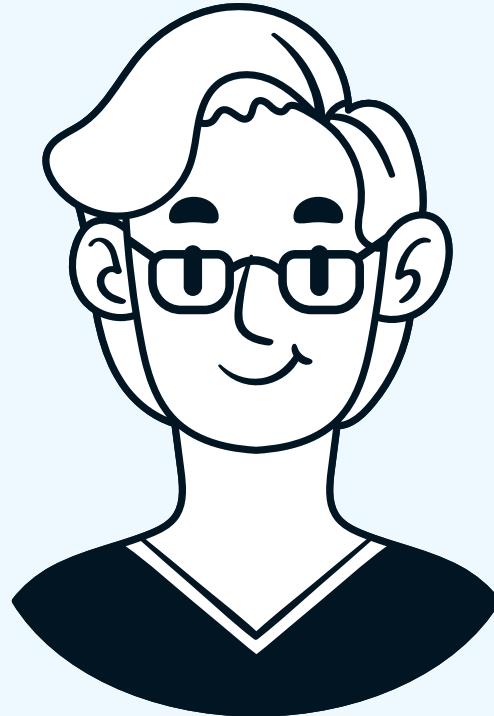
#디지털기기사용

#취미는운동

#섬세함

1. Persona

페르소나



교육용 서버 운영 중인 외국인 강사 - Big Dragon (30세)

“학생들이 교육용 서버에 잘 접속하고 있는지 접속 여부를 쉽게 확인하고 싶어. 매번 서버에 직접 들어가서 확인하는 건 너무 불편하고, 검은 화면에 하얀색 글씨는 직관적이지 않아.”

- 서버 접속 없이도 실시간 접속자 현황을 쉽게 확인하고 싶음
- 텍스트 기반 인터페이스 대신 그래픽 기반의 직관적인 UI 필요

#체계적

#취미는독서

#직관성을중시

1. Persona

페르소나



서버 초보 운영자 – 서윤배 (25세, 대학생)

“서버를 만들어서 서비스까지 가동했는데, 서버에서 무슨 일이 일어나는지 잘 모르겠어. 어떤 자원들이 있는지, 네트워크가 어떻게 처리되는지 직관적으로 보고 싶은데 방법이 없을까?”

- 서버 및 네트워크 구성 요소를 쉽게 이해할 수 있는 시각화 제공
- 현재 서버에서 어떤 프로세스가 실행 중인지 확인할 수 있는 기능 필요

#초보개발자

#취미는코딩

#코딩조아

Timeline

일정

2. Timeline

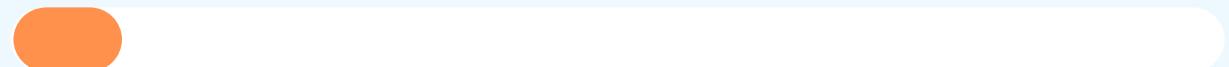
일정

2025.02.13 ~ 2025.02.27

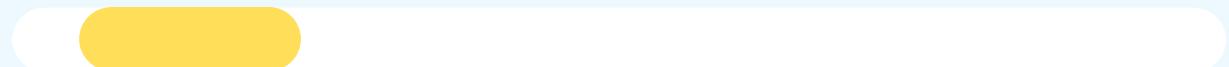
작업기간 : 총 15일

13 15 17 19 21 23 25 27

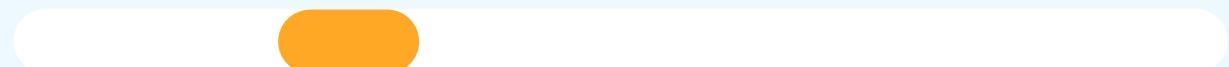
요구사항 분석



기능명세서 및 화면설계서 작성



개발환경 설정



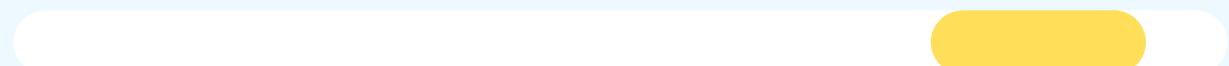
Front-end 개발 및 Back-end 개발



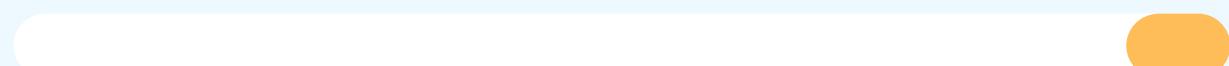
Front-end 개발 및 Back-end 개발



기능 개선 및 보완 작업



문서화

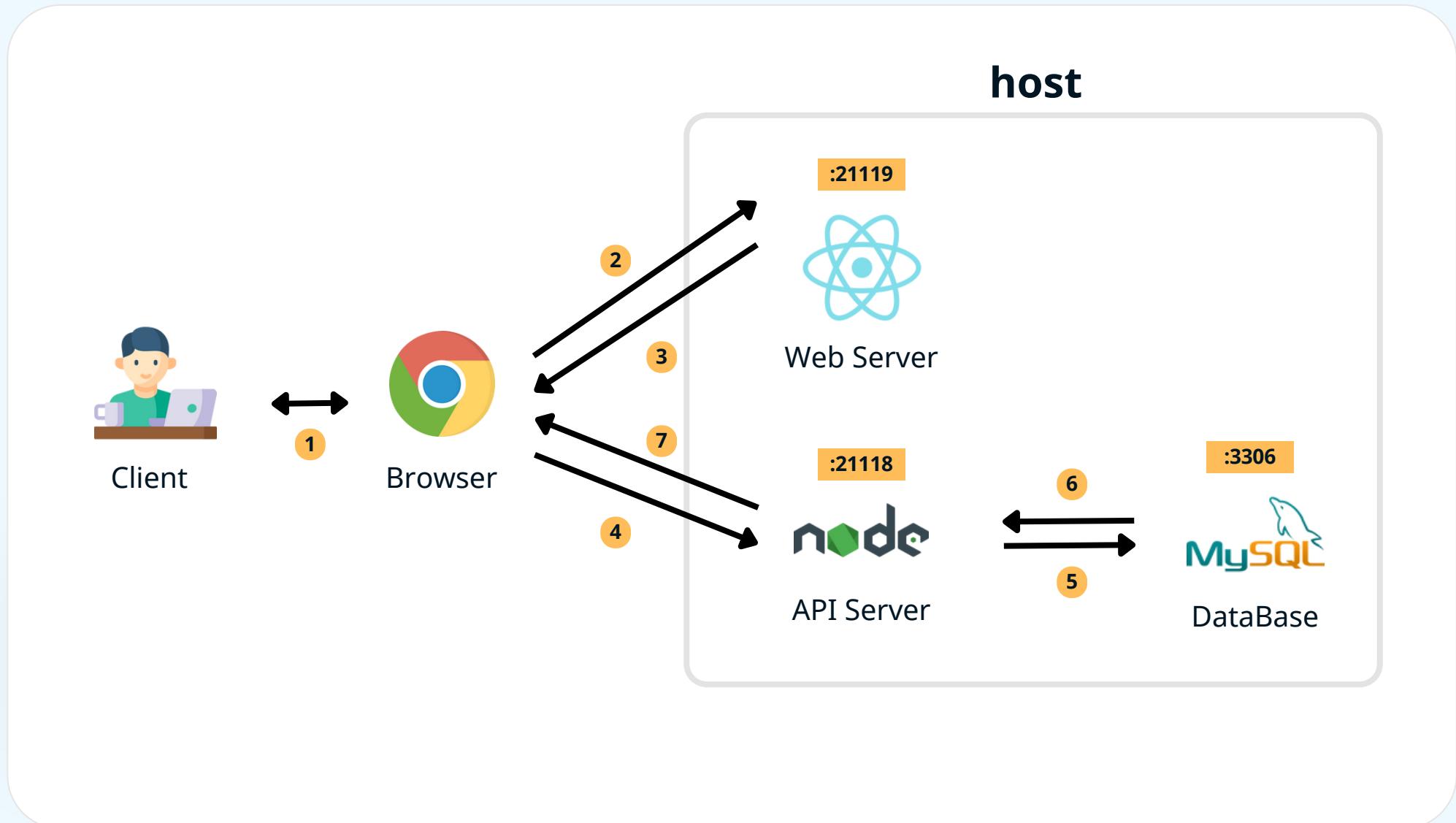


Stack

기술

3. Stack

시스템 아키텍쳐



3. Stack

활용 기술



3. Stack

리눅스 명령어

ifconfig

네트워크 인터페이스 정보 확인 (송·수신 패킷 수)

df

디스크 사용량 및 여유 공간 확인

top

현재 CPU 및 SWAP 메모리 사용량 실시간 모니터링

lscpu

CPU 처리 속도 확인

ping

네트워크 응답 시간 측정

last

최근 로그인한 사용자 목록 및 접속 기록 확인

ss

현재 사용 중인 포트의 연결 수 확인

netstat

현재 사용중인 포트 확인

Wireframe

화면 설계

4. Wireframe

화면 설계

SCREEN NAME	Main	SCREEN LOCATION	Home
DashBoard Main			
	 s1	 s2	 s3
	 2	 2	 2
		 전체통계보기	 2
			 s4

1 버튼

해당하는 각 페이지로 이동하는 버튼입니다.

2 그래프

각 서버들의 중요한 데이터를 시각적으로 실시간 모니터링 가능한 모듈입니다.

클릭 시 각 서버별 상세보기(Server Detail) 페이지로 이동합니다.

4. Wireframe

화면 설계

SCREEN NAME Detail **SCREEN LOCATION** Server Detail

DashBoard Detail - s1

Home
S1
S2
S3
S4

런타임
가장 많이 이용하는 포트
활성 포트 목록

전체통계보기
주간통계보기
인터넷 응답 시간
수신 패킷량
송신 패킷량
서버 접속 유저 목록

- 1 버튼**
해당하는 각 페이지로 이동하는 버튼입니다.
- 2 시간**
서버가 동작한 시간을 나타냅니다.
- 3 그래프**
한눈에 알아보기 쉽도록 그래프로 시각화합니다.
- 4 표**
목록들을 정리하여 보기 쉽게 표현하는 표 형식의 리스트입니다.

4. Wireframe

화면 설계

SCREEN NAME	Total	SCREEN LOCATION	Server Total
Home			
S1	Dashboard Total - s1	1 조회기간	2 전체통계보기
S2	3 메모리 평균 사용량	2 실시간 모니터링	
S3	3 송수신 패킷량	3 스왑 메모리 평균 사용량	3 CPU 평균 사용량
S4	4 서버 원격 접속 로그	3 서버 방문자수	3 에러 발생수
		4 에러 로그	

- 1 날짜 선택 모듈**
기간을 설정하는 기능입니다.
- 2 버튼**
해당하는 각 페이지로 이동하는 버튼입니다.
- 3 그래프**
한눈에 알아보기 쉽도록 그래프로 시각화합니다.
- 4 표**
목록들을 정리하여 보기 쉽게 표현하는 표 형식의 리스트입니다.

4. Wireframe

화면 설계

SCREEN NAME All total **SCREEN LOCATION** Server All Total

The wireframe displays the 'Dashboard Main' screen with the following layout:

- Top Left:** A vertical sidebar with circular buttons labeled Home, S1, S2, S3, and S4.
- Top Right:** Two buttons labeled 1 (조회기간) and 2 (메인페이지).
- Middle Left:** A large box labeled 3 (메모리 사용량 비교).
- Middle Right:** A large box labeled 3 (CPU 사용량 비교).
- Bottom Left:** A large box labeled 3 (패킷 송수신량 비교).
- Bottom Right:** A large box labeled 3 (접속량 비교).

1 날짜 선택 모듈

기간을 설정하는 기능입니다.

2 버튼

해당하는 페이지로 이동하는 버튼입니다.

3 그래프

한눈에 알아보기 쉽도록 그래프로 시각화합니다.

UI design

UI 디자인

5. UI design

UI 디자인 - Server overview



반응형 네트워크 대시보드 - 어디서나 접근 가능한 모니터링 솔루션

현대적인 웹 기술을 활용하여 완벽한 반응형 디자인을 구현하였습니다. 이를 통해 사용자는 일반 PC, 모바일 등의 다양한 디바이스에서 일관된 경험을 누릴 수 있습니다.

5. UI design

UI 디자인 - Colors & Rounding box

#010101

#061c31

#9933fd

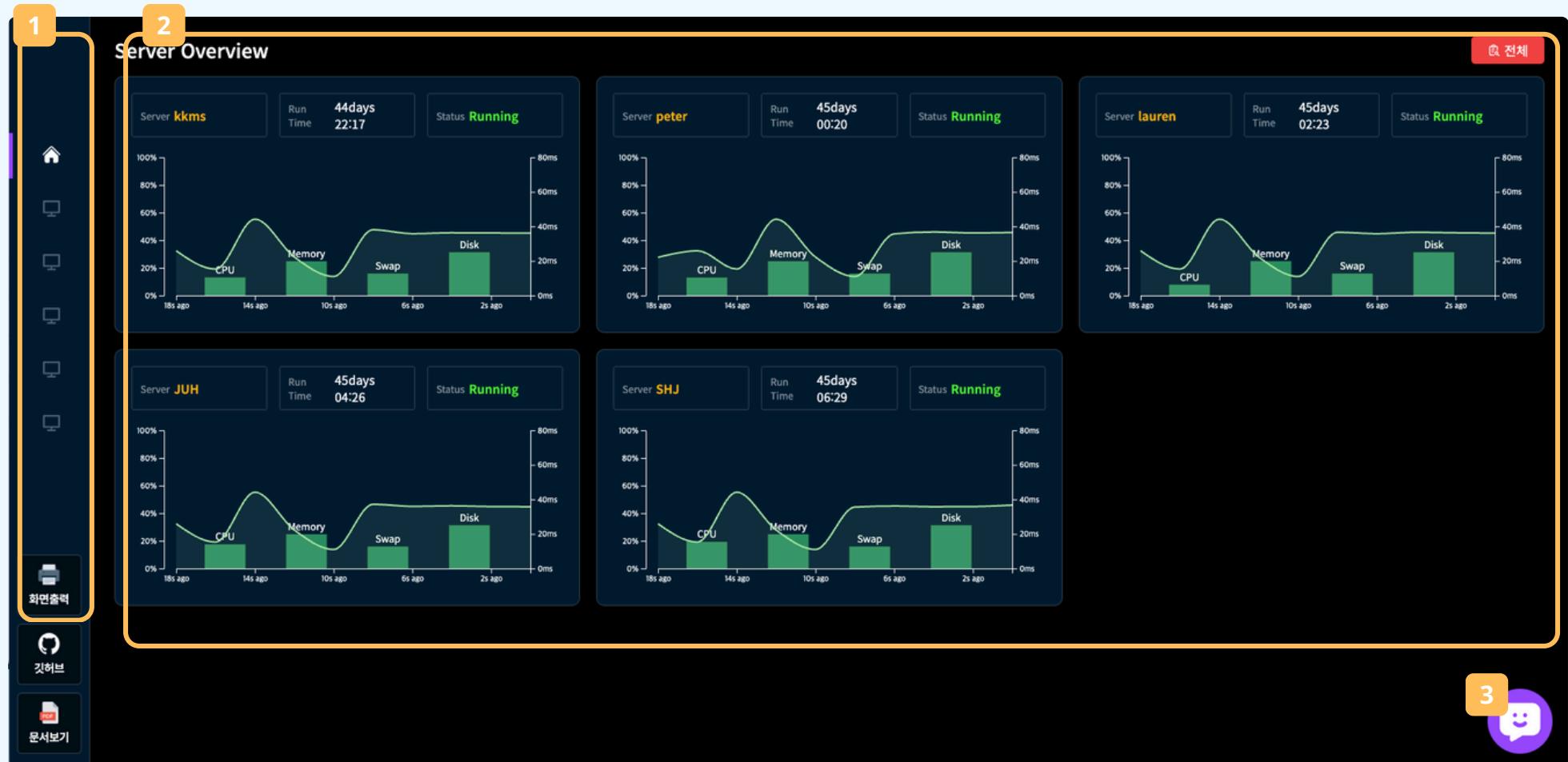
정보의 가독성을 향상, 대시보드의 전문성과 기능성을 강조

눈의 피로를 줄이고 장시간 모니터링에 적합한 다크모드 디자인을 하였으며, 중요한 데이터를 즉각적으로 식별할 수 있도록 강조색을 적절히 사용하였습니다.

따라서 사용자가 편안하게 모니터링 할 수 있으며, 효율적으로 서비스를 이용할 수 있도록 돋습니다.

5. UI design

UI 디자인 - Server overview



1 서버 네비게이션

각 서버를 대표하는 아이콘을 세로로 배열하였으며, 선택된 서버가 하이라이트 되어 시안성을 높임

2 서버 상태 표시 영역

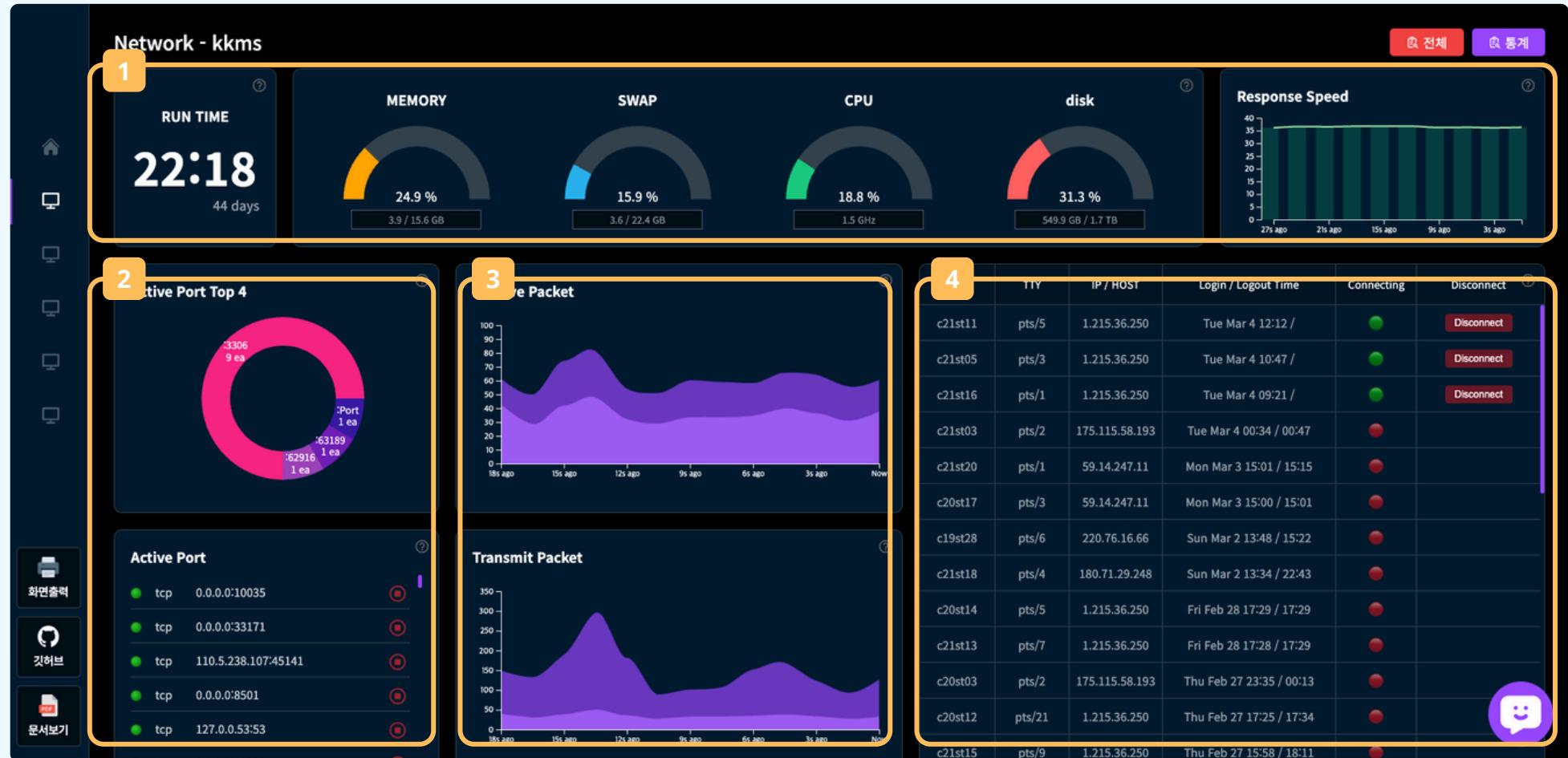
현재 관리하는 5개의 서버를 동시에 확인할 수 있는 영역으로, 그리드 레이아웃을 이용한 사용자 친화적 디자인

3 챗봇 서비스

우측 하단에 챗봇 서비스를 사용할 수 있도록 아이콘 고정 출력

5. UI design

UI 디자인 - Network Dashboard



1 서버 상태 모니터링

서버 런타임, 메모리나 CPU 등의 주요 상태를 쉽게 알아볼 수 있도록 시각화하여 출력

2 포트 모니터링

활성 포트를 리스트로 나열하고, 상위 4 개 포트의 비율을 직관적으로 확인할 수 있도록 도넛 그래프로 표시

3 실시간 패킷 사용량

초당 송수신되는 패킷양을 눈에 띄는 컬러를 이용해 실시간 그래프로 표시해 빠르게 파악할 수 있도록 디자인

4 접속 유저 관리

테이블 형식의 디자인을 이용해 접속 유저의 정보 출력 뿐 아니라, 특정 유저의 접속을 끊는 기능 지원

5. UI design

UI 디자인 - Total



1 datepicker를 이용한 달력 UI

날짜를 직관적으로 선택할 수 있는 달력 인터페이스를 사용했으며, 반응형으로 다양한 디바이스에서 동일한 사용성 제공

2 기간별 사용량 그래프화

그래프의 부드러운 곡선과 전체적으로 통일감 있는 채도를 사용하여 데이터를 시각적으로 매끄럽게 표현

5. UI design

UI 디자인 - Server Insight



1 데이터 해석을 돋는 범례

사용자가 그래프를 해석할 때 빠르게 정보를 파악 할 수 있도록 서버별 색상 정보를 한번에 범례로 만들어 표기

2 datepicker를 이용한 달력 UI

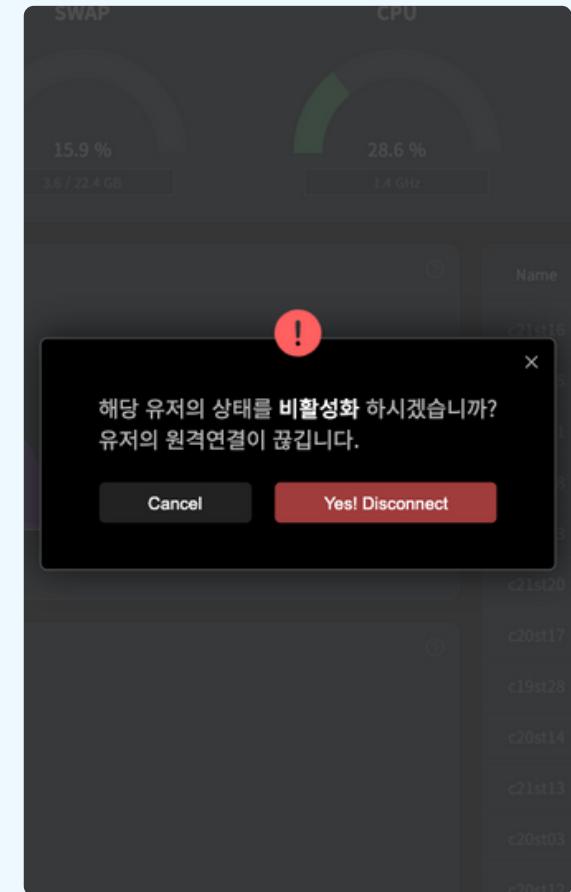
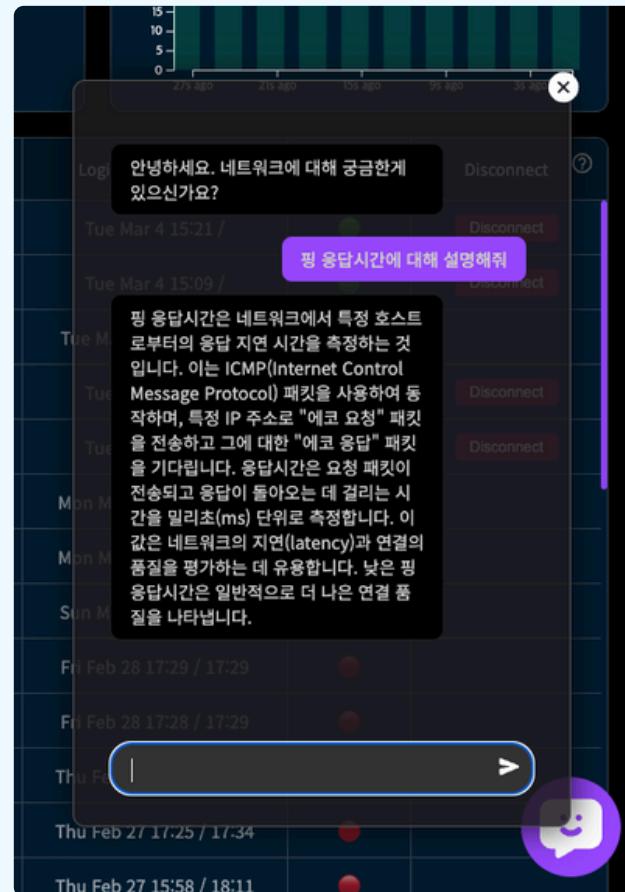
날짜를 직관적으로 선택할 수 있는 달력 인터페이스를 사용했으며, 반응형으로 다양한 디바이스에서 동일한 사용성 제공

3 서버별 사용량 그래프

서버별 사용량을 통일된 색상으로 표현해 사용자의 데이터 해석이 편리하도록 도움

5. UI design

UI 디자인 - Server Insight



1 어디서나 쉽게 접근할 수 있는 챗봇

모든 페이지 하단에 있는 아이콘을 눌렀을 때, 챗봇 서비스 이용 가능, 눈의 피로를 줄여 사용자가 대화에 집중할 수 있도록 돋는 다크모드 디자인.

2 손쉬운 사용을 돋는 도움말

요소의 우측 상단에 있는 물음표 아이콘을 눌러 각 요소가 나타내는 정보에 대한 간략한 설명을 확인할 수 있도록 표시

3 모달창을 이용한 유연한 상호작용

사용자에게 신뢰감을 주는 깔끔하고 직관적인 디자인의 모달창 디자인으로 유연한 상호작용 가능

Code Analysis

코드 분석

6. Code Analysis

개발 환경



```

version: '3.0'
services:
  express-app:
    image: peterseuo9503/network_back_end // Ubuntu, node, express 가 미리 깔려있는 이미지
    container_name: express-container
    working_dir: /app
    volumes:
      - ./app // 도커 컴포즈 파일이 있는 호스트 폴더와 볼륨 설정
    ports:
      - "3000:3000" // 개발환경 포트번호
    environment:
      - DB_HOST=...
      - DB_USER=...
      - DB_PASSWORD=...
      - DB_NAME=...
    depends_on:
      - mariadb
    stdin_open: true
    tty: true
  mariadb:
    image: mariadb:10.3.39 // 실제 적용되는 서버와 동일한 버전
    container_name: mariadb-container
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ...
      MYSQL_DATABASE: network
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
  volumes:
    mysql-data:

```

docker-compose.yml

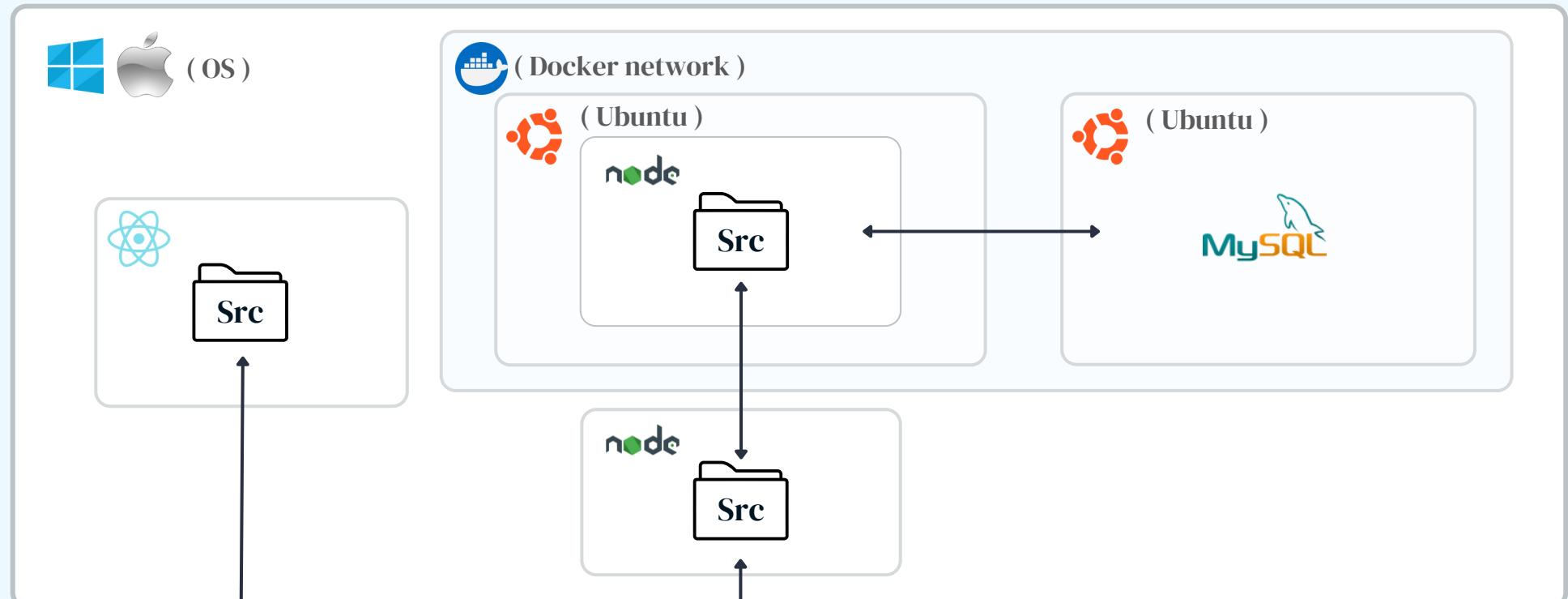
개발환경을 위한 Docker 세팅

- Node.js 기반 Express 컨테이너 실행
- 호스트의 코드를 앱 /app에 마운트
- DB 컨테이너를 실행

6. Code Analysis

개발 환경

Local



Git





Front end

6. Code Analysis

프론트엔드 소스 파일 구조

전체 구조

```
src
├── assets
│   ├── config
│   │   └── url.ts
│   ├── favicon.png
│   ├── github.png
│   ├── link.jpg
│   ├── pdf.png
│   └── printer.png
├── types
│   ├── forTotal
│   │   └── forTotal.ts
│   └── index.ts
└── components
    ├── styles
    │   ├── App.css
    │   ├── App.tsx
    │   └── index.css
```

components

```
components
├── Dashboard-main.tsx
│   └── Dashboard-mainbox.tsx
├── Dashboard-detail.tsx
│   └── ...
├── Dashboard-total.tsx
│   └── ...
├── Dashboard-all-total.tsx
│   └── ...
├── Nav-bar.tsx
│   └── ...
├── Chatbot.tsx
│   └── Chatbot-chatting.tsx
```

styles

```
styles
├── _mixin.scss
├── _variables.scss
├── all-total-...
├── board-...
├── chatbot-...
├── total-...
├── dashboard-all-total.module.scss
├── dashboard-detail.module.scss
├── dashboard-main.module.scss
└── dashboard-total.module.scss
└── nav-bar.module.scss
```

6. Code Analysis

프론트엔드 코드분석



Sass

```

1 @mixin box-item {
2   position: relative;
3   background-color: variables.$background-Boxes;
4   border: 1px solid variables.$border-Boxes;
5   padding: 20px;
6   border-radius: 10px;
7   overflow: hidden;
8 }
9 @mixin pagetitle {
10   font-size: 24px;
11   font-weight: 500;
12   span {
13     color: #FBB214;
14   }
15   @media (max-width: 550px) {
16     font-size: 18px;
17   }
18 }
19 @mixin title {
20   text-align: center;
21   font-weight: bold;
22   font-size: 17px;
23   @media (max-width: 550px) {
24     margin-bottom: 10px;
25   }
26 }
27

```

_mixin.scss

```

1 @use 'variables';
2 @use 'mixin';
3
4 .body {
5   @include mixin.box-item;
6   flex: 1;
7
8   .title {
9     @include mixin.title;
10    & { /*yoon: 오류발생으로 명시적 선언합니다*/
11      text-align: left;
12    }
13
14   .helpBtn {
15     @include mixin.helpBtn;
16   }
17
18   .helper {
19     @include mixin.helper;
20   }
21 }
22
23

```

total-connect.module.scss

Sass를 이용하여 효율적인 스타일 관리

- CSS 확장 언어 Sass 활용
- 코드 재사용성 극대화 및 스타일 관리 효율성 향상
- 변수, 믹스인, 중첩, 상속 등의 기능 활용
- 유지보수성 향상

6. Code Analysis

프론트엔드 코드분석



Sass

```

1  @media (max-width: 1500px) {
2      grid-column: 1 / span 2;
3      height: 230px;
4      order: 3;
5  }
6  @media (max-width: 920px) {
7      display: grid;
8      grid-template-columns: 1fr 1fr;
9      height: auto;
10     align-items: center;
11     gap: 50px 10px;
12     padding-bottom: 50px;
13 }
14 @media (max-width: 440px) {
15     grid-template-columns: 1fr;
16     gap: 40px;
17 }
18 .title {
19     margin-bottom: 0px;
20     margin-top: 5px;
21     @include mixin.title;
22 }
23 .memory {
24     position: relative;
25     width: 25%;
26     @media (max-width: 920px) {
27         width: 80%;
28         height: 170px;
29         margin: 0 auto;
30     }
31 }           _mixin.scss

```

```

1  .btngroup {
2      display: flex;
3      gap: 10px;
4      .alltotal {
5          background-color: variables.$all-total-color;
6      }
7  }
8  a {
9      display: flex;
10     padding: 5px 20px;
11     background-color: variables.$primary-color;
12     color: #fff;
13     text-decoration: none;
14     border-radius: 5px;
15     gap: 5px;
16     align-items: center;
17     @media (max-width: 550px) {
18         padding: 5px 10px;
19         font-size: 12px;
20         margin-top: 3px;
21     }
22 }

```

total-connect.module.scss

다양한 디바이스를 지원하는 반응형 디자인

- 미디어 쿼리를 활용한 반응형 디자인
- 다양한 디바이스 적응성 증대

6. Code Analysis

프론트엔드 코드분석

TS Typescript

```

1  export interface CpuData {
2    date: string;
3    value: number;
4  }
5
6  export interface PacketData {
7    date: string;
8    rxData: number;
9    txData: number;
10 }
11
12 export interface WebConnectData {
13   date: string;
14   value: number;
15 }
16
17 export interface ErrGraphData {
18   date: string;
19   web: number;
20   ufw: number;
21   auth: number;
22   mysql: number;
23 }
24
25 export interface LoginData {
26   server_id: number;
27   user: string;
28   login_count: number;
29   last_login_time: string;
30 }

```

forTotal.ts

```

1  export interface DashboardAllTotalProps {
2    serverName: string;
3    setPage: () => void;
4    goAllTotal: ()=> void;
5  }
6
7  export interface MemoryData {
8    date: string;
9    value: number;
10 }
11 export interface MemorySwapData {
12   date: string;
13   value: number;
14 }
15
16 export interface ServerReverseMapping {
17   [key: string]: number;
18 }
19
20 export interface AllTotalPageData {
21   recorded_date: string; // ISO 형식의 날짜 문자열
22   server_id: number;
23   mem_avg: string;
24   cpu_avg: string;
25   rx_data: string;
26   tx_data: string;
27   total: string;
28   web_access_count: number;
29   web_error_count: number;
30   ufw_count: number;
31   auth_error_count: number;
32   mysql_err_count: number;
33   critical_cnt: number | null;
34 }

```

forTotal.ts

안정성과 효율성을 갖춘 타입스크립트 활용

- 타입 정보를 통한 코드 의도 명확화
- 안정적인 코드 구현을 돋는 정적 타입 검사
- 여러 개발자의 작업에서 코드 일관성 유지 및 리팩토링 시 발생되는 오류 최소화

6. Code Analysis

프론트엔드 코드분석

React

TypeScript

```

1  useEffect(() => {
2    const getData = (data: { [key: string]: string }[], serverName: string, type: string) => {
3      const server = data.find((serverData) => serverName in serverData);
4      if (!server || !(serverName in server)) {
5        return undefined;
6      }
7      return server[serverName][type];
8    }
9
10   const fetchData = async (path: string) => {
11     const res = await fetch(`${url.url}/${path}`);
12     return res.json();
13   }
14
15   const displayServers = async () => {
16     const status = await fetchData("status");
17     const runtime = await fetchData("runtime");
18     const ping = await fetchData("ping");
19
20     if (status && runtime && ping) {
21       setServerBody(serverList.map((server, index) => {
22         return (
23           <div
24             key={index}
25             className={style.body}
26             onClick={() => {
27               setPage("detail");
28               changeServer(server);
29             }}
30           >
31             <DashboardMainBox
32               key={server}
33               server={server}
34               runtime={getData(runtime, server, "runtime")}
35               memory={(Number(getData(status, server, "memory"))["usingMemory"] * 100 / Number(getData(status, server, "memory"))).toFixed(2)}
36               swap={(Number(getData(status, server, "swap"))["usingSwap"] * 100 / Number(getData(status, server, "swap"))).toFixed(2)}
37               cpu={getData(status, server, "cpu")["cpuUtilization"]}
38               disk={(Number(getData(status, server, "disk"))["usingDisk"] * 100 / Number(getData(status, server, "disk"))).toFixed(2)}
39               ping={getData(ping, server, "pingResponse")}
40             </DashboardMainBox>
41           </div>
42         );
43       }));
44     }
45   });

```

데이터 가공

비동기 데이터 가져오기

상태 업데이트

Dashboard-main.tsx

비동기 데이터 처리

- 실시간 데이터 반영 (useEffect)
- 비동기 데이터 가져오기 (fetchData)
- 데이터 가공 및 처리 (getData)
- 상태 업데이트

6. Code Analysis

프론트엔드 코드분석

React

TypeScript

D3.js

```

1  useEffect(() => {
2      if (!boxGraph.current) return;
3
4
5      // SVG 설정
6      const svg = d3.select(boxGraph.current)
7          .attr("width", width)
8          .attr("height", height);
9
10     svg.selectAll("*").remove();
11
12     // 막대 그래프 데이터
13     const barData = [
14         { name: "CPU", value: parseFloat(cpu) },
15         { name: "Memory", value: parseFloat(memory) },
16         { name: "Swap", value: parseFloat(swap) },
17         { name: "Disk", value: parseFloat(disk) }
18     ];
19
20     // 스케일 설정
21     const xScaleBar = d3.scaleBand()
22         .domain(barData.map(d => d.name))
23         .range([0, innerWidth])
24         .padding(0.5);
25
26     const yScaleBar = d3.scaleLinear()
27         .domain([0, 100])
28         .range([innerHeight, 0]);
29
30     const yScalePing = d3.scaleLinear()
31         .domain([0, Math.max(...pingHistory, 80)])
32         .range([innerHeight, 0]);
33

```

Dashboard-mainbox.tsx

```

// 막대 그래프 그리기
graphGroup.selectAll(".bar")
    .data(barData)
    .enter()
    .append("rect")
        .attr("class", "bar")
        .attr("x", d => xScaleBar(d.name) || 0)
        .attr("y", d => yScaleBar(d.value))
        .attr("width", xScaleBar.bandwidth())
        .attr("height", d => innerHeight - yScaleBar(d.value))
        .attr("fill", d => {
            const value = d.value;
            if (value < 50) return "rgba(74, 222, 128, 0.6)";
            if (value < 75) return "rgba(251, 191, 36, 0.6)";
            return "rgba(239, 68, 68, 0.6)";
        });
// 막대 그래프 레이블
graphGroup.selectAll(".bar-label")
    .data(barData)
    .enter()
    .append("text")
        .attr("class", "bar-label")
        .attr("x", d => (xScaleBar(d.name) || 0) + xScaleBar.bandwidth() / 2)
        .attr("y", d => yScaleBar(d.value) - 5)
        .attr("text-anchor", "middle")
        .text(d => `${d.name}`)
        .attr("fill", "white")
        .attr("font-size", "12px");
[cpu, memory, swap, disk, pingHistory]);

```

Dashboard-mainbox.tsx

D3.js를 이용한 데이터 시각화

- JSON 데이터를 비동기로 가져오기 위해 fetch 사용
- props를 이용한 효율적 데이터 전달
- D3를 이용한 차트, 그래프, 기타 데이터 시각화 요소 렌더링

Server Overview



6. Code Analysis

프론트엔드 코드분석

React

TypeScript

D3.js

```

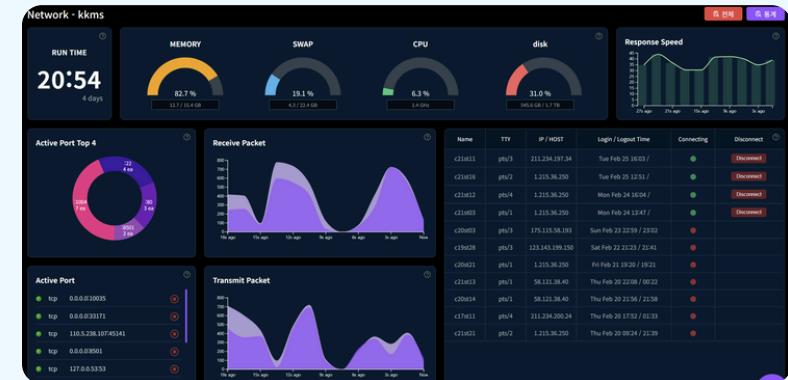
1  const DashboardDetail: React.FC<ParentProps> = ({ serverName, setPage, goAllTotal }) => {
2      return <div className={style.dashboard}>
3          <div className={style.header}>
4              <h1 className={style.title}>Network - {serverName}</h1>
5              <div className={style.btnGroup}>
6                  <a href="#" className={style.alltotal}>
7                      onClick={e=>{
8                          e.preventDefault();
9                          goAllTotal();
10                     }}
11                  >TbReportSearch /> 전체</a>
12                  <a href="#" onClick={ e=>{
13                      e.preventDefault();
14                      setPage();
15                  }}>TbReportSearch /> 통계</a>
16              </div>
17          </div>
18          <div className={style.board}>
19              <div className={style.boardTop}>
20                  <BoardRuntime serverName={serverName}></BoardRuntime>
21                  <BoardStatus serverName={serverName}></BoardStatus>
22                  <BoardResspeed serverName={serverName}></BoardResspeed>
23              </div>
24              <div className={style.boardBody}>
25                  <BoardPorts serverName={serverName}></BoardPorts>
26                  <BoardPacket serverName={serverName}></BoardPacket>
27                  <BoardUsers serverName={serverName}></BoardUsers>
28              </div>
29          </div>

```

Dashboard-detail.tsx

React 컴포넌트 기반 구조

- 재사용성과 유지보수성이 높은 리액트 컴포넌트 이용
- 독립적 컴포넌트를 조합한 효율적 개발 진행



6. Code Analysis

프론트엔드 코드분석

React

TypeScript

```

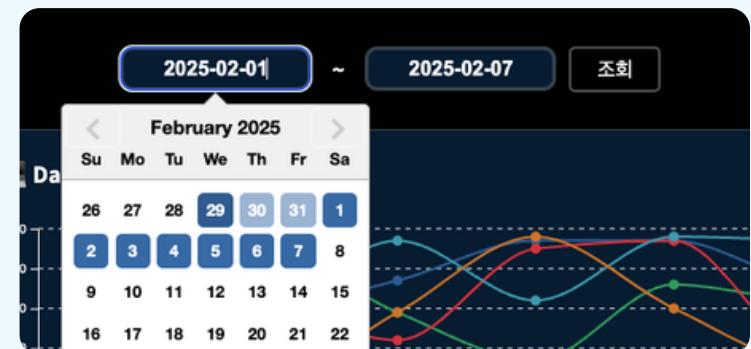
1  const handleStartDateChange = (date: Date) => {
2    //일주일 범위 유지를 위한 코드
3    const oneWeekLater = new Date(date?.getTime() + 6 * 24 * 60 * 60 * 1000);
4
5    if (endDate < date || endDate > oneWeekLater) {
6      setEndDate(oneWeekLater);
7      setStartDate(date);
8      onDateChange(date, oneWeekLater);
9    } else {
10      setStartDate(date);
11      onDateChange(date, endDate);
12    }
13  };
14
15 const handleEndDateChange = (date: Date) => {
16  //일주일 범위 유지를 위한 코드
17  const oneWeekBefore = new Date(date.getTime() - 6 * 24 * 60 * 60 * 1000);
18  if (startDate > date || startDate < oneWeekBefore) {
19    setStartDate(oneWeekBefore);
20    setEndDate(date);
21    onDateChange(oneWeekBefore, date);
22  } else {
23    setEndDate(date);
24    onDateChange(startDate, date);
25  }
26};

```

[All-total-datepicker.tsx](#)

통계페이지 날짜별 조회 구현

- 외부 라이브러리(react-datepicker)를 이용한 날짜 선택 UI 구현
- 사용자가 선택한 날짜를 7일로 제한하여 자동 변경되도록 세팅





Back end

6. Code Analysis

백엔드 소스 파일 구조

Express

```

📦src
  ├─ config
  |  └─ daemon.js
  ├─ database
  |  └─ info.js
  |  └─ queries.js
  ├─ route
  |  ├─ module
  |  |  └─ readFile.js
  |  └─ chat_bot.js
  |  └─ get_total_all_page.js
  |  └─ get_total_page_info.js
  |  └─ route_JSON.js
  └─ app.js

```

실시간 데이터

```

📦server
  ├─ json
  |  ├─ activePort.json
  |  ├─ ifconfig.json
  |  ├─ manual.txt
  |  ├─ ping.json
  |  ├─ runtime.json
  |  ├─ server.json
  |  ├─ status.json
  |  └─ usedPort.json
  └─ userList.json
  └─ command.js
  └─ server.js

```

통계 데이터

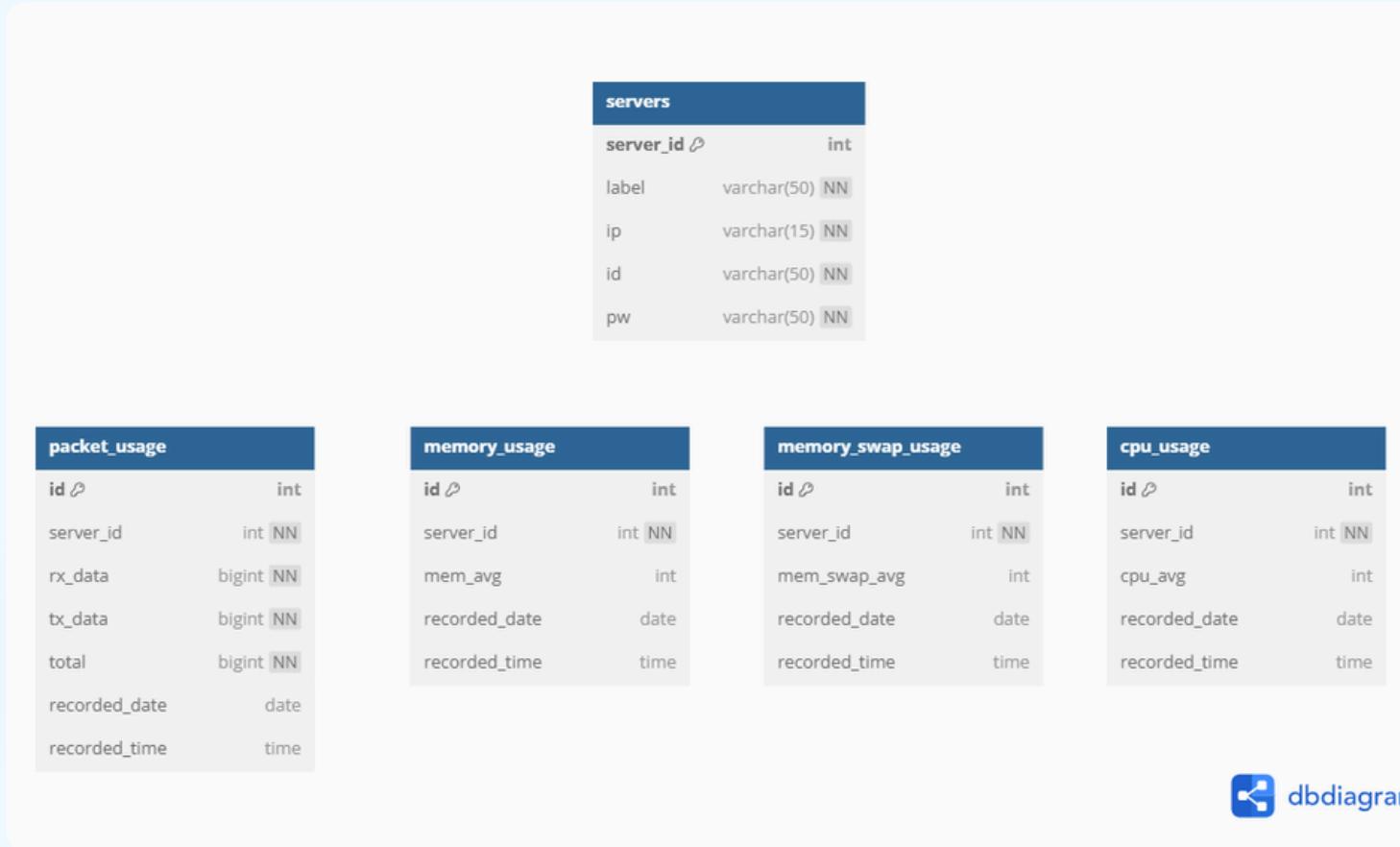
```

📦db_init
  ├─ logs
  |  └─ ...
  ├─ log_manager
  |  └─ initTable.js
  |  └─ insertDumyData.js
  |  └─ LogModuleRunner.js
  ├─ log_modules
  |  └─ ...
  ├─ sql
  |  └─ Database_tables.sql
  |  └─ dumy_data_insert.sql
  |  └─ sqlTemplate.js
  ├─ utils
  |  └─ ...
  └─ app.js
  └─ InsertStatistics.js

```

6. Code Analysis

데이터베이스 테이블



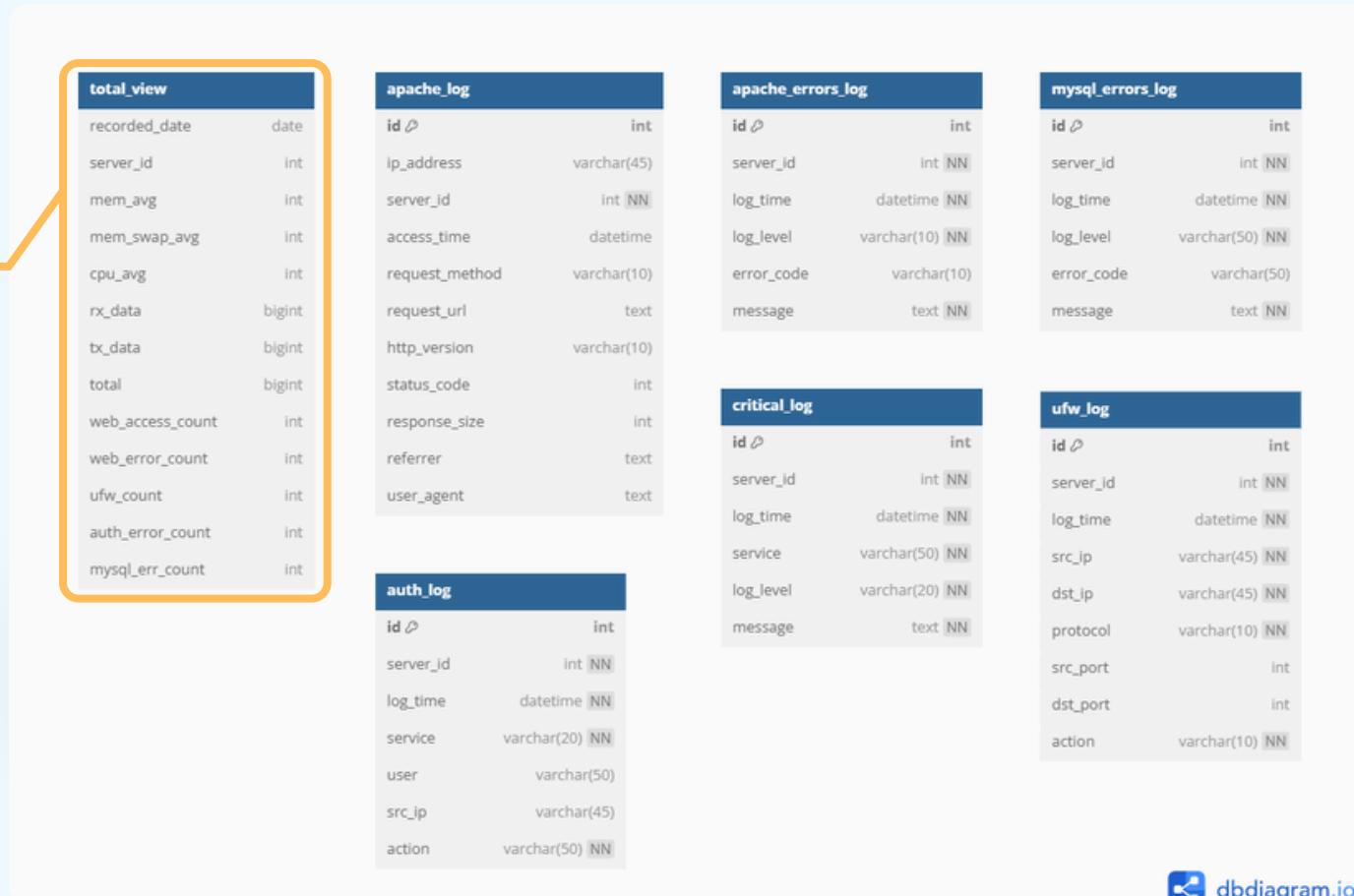
서버 정보 테이블 및 실시간 정보 데이터 수집 테이블

- 모든 테이블은 서버 구분 정보를 담은 servers 테이블로 관리
- 30분 간격 누적 값을 기록 하는 usage 테이블

6. Code Analysis

데이터베이스 테이블

단일 통계 뷰



dbdiagram.io

모든 데이터를 한눈에 확인할 수 있는 단일 통계 뷰와 로그테이블

- 우분투 로그와 유사한 데이터를 저장하는 로그 테이블
- 데이터 조회를 용이하게 하기위해 로그테이블과 usage 테이블과 결합해 단일 통계 뷰를 구성

6. Code Analysis

백엔드 코드분석

node Node.js

MySQL

```

1  class AuthLogProcessor {
2      constructor() {...}
3
4      async readFile() {
5          const filePath = path.join(__dirname, "../logs/auth_log.txt");
6          try {} catch (err) {...}
7      }
8
9      parseLogs(logs) {
10         if (!logs) {
11             console.error("로그 데이터가 없습니다.");
12             return [];
13         }
14         const parsingLines = logs.trim().split(/\n/).map(line => line.replace(/\r$/));
15         const insertValues = parsingLines.map(line => {
16             const matchLog = line.match
17                 (/^(.*?) \[SERVICE: (\d+) \] \[USER: (.*)\] \[SRC: (.*)\] \[ACTION: (.*)\]/);
18             if (!matchLog) return null;
19             const log_time = matchLog[1];
20             const server_id = matchLog[2];
21             const service = matchLog[3];
22             const user = matchLog[4];
23             const src_ip = matchLog[5];
24             const action = matchLog[6];
25             return [log_time, server_id, service, user, src_ip, action];
26         }).filter(v => v !== null);
27         return insertValues;
28     }
29
30     async insertLogs(insertValues) {...}
31
32     async run() {
33         try {
34             const fileContent = await this.readFile();
35             const values = this.parseLogs(fileContent);
36             await this.insertLogs(values);
37         } catch (err) {...}
38     }
39 }
40
41 module.exports = AuthLogProcessor;
42

```

AuthLogProcessor.js.js

로그 파싱 클래스

- 로그 규칙 분석 및 파싱
(문자열, 배열 고차함수, 정규식 활용)
- 파싱 결과 데이터베이스 저장

6. Code Analysis

백엔드 코드분석

node Node.js

MySQL

```

1 const si = require('systeminformation');
2
3 class InsertStatistics {
4   constructor() {...}
5
6   async getStatistics() {
7     try {
8       // 메모리 측정 및 평균 계산
9       const mem = await si.mem();
10      this.mem_usage += ((mem.total - mem.free) / mem.total) * 100;
11      this.mem_avg = this.mem_usage / this.flag;
12      // 스왑 메모리 사용량 측정
13      this.mem_swap_usage += (mem.swapused / mem.swaptotal) * 100;
14      this.mem_swap_avg = this.mem_swap_usage / this.flag;
15      // CPU 사용량 측정 및 평균 계산
16      const cpus = await si.currentLoad();
17      this.cpu_usage += cpus.currentLoad;
18      this.cpu_avg = this.cpu_usage / this.flag;
19      console.log("cpu_avg :", this.cpu_avg);
20      // 네트워크 데이터 측정
21      const interfaces = await si.networkInterfaces();
22      const openIfaces = interfaces.filter(itf => itf.operstate === "up");
23      const ifaceNames = openIfaces.map(itf => itf.iface);
24      this.flag++;
25    } catch (err) {...}
26  }
27
28   async insertDB() {...}
29
30   start() {
31     this.statisticsInterval = setInterval(() => {
32       this.getStatistics();
33     }, 2000);
34     // 30분마다 DB 삽입 실행
35     this.insertInterval = setInterval(() => {
36       this.insertDB();
37     }, 1800000);
38   }
39 }

```

InsertStatistics.js

누적 데이터 삽입 클래스

- Node.js Systeminformation 모듈을 활용하여 서버 메모리, CPU, 네트워크 데이터 수집
- 통계용 데이터를 30분 동안 누적하고 초 단위 평균 계산
- 30분마다 누적 데이터를 DB에 저장하고 초기화 후 다시 수집 시작

6. Code Analysis

백엔드 코드분석

node Node.js



```
1 const LogModuleRunner = require("./log_manager/LogModuleRunner");
2 const InsertStatistics = require("./InsertStatistics");
3 const runner = new LogModuleRunner();
4 const insertStatistics = new InsertStatistics();
5 runner.run();
6 insertStatistics.start();
```

app.js

통합 데이터 수집 및 DB 설정 자동화 파일

- SQL 실행과 데이터 입력, 실시간 데이터 수집 기능을 통합한 app.js
- 파일 실행 시 DB 설정 자동 구성 및 주기적 시스템 정보 수집 진행

6. Code Analysis

백엔드 코드분석

node Node.js

```

1  async ping() {
2      try {
3          const { stdout, stderr } = await this.exec("ping -c 1 8.8.8.8");
4          return stdout.trim();
5      } catch (err) {
6          const randomPing = (this.randomReturner(30, 45).toFixed(1));
7          // console.log(randomPing);
8          return (
9              PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
10             64 바이트 (8.8.8.8에서): icmp_seq=1 ttl=56 시간=${randomPing} ms
11
12             --- 8.8.8.8 펑 통계 ---
13             1 패킷이 전송되었습니다, 1 수신되었습니다, 0% 패킷 손실, 시간 0ms
14             rtt 최소/평균/최대/표준편차 = 35.221/35.221/35.221/0.000 ms
15         `).trim();
16     }
17 }
```

command.js

데이터 가져오는 메서드

- exec 함수를 사용하여 데이터를 가져오는 방식은 직접 명령어를 실행해 필요한 정보를 불러오는 구조
- 오류가 발생하거나 명령어를 실행할 수 없는 환경에서는, 실제 명령어와 동일한 형태의 문자열을 반환

6. Code Analysis

백엔드 코드분석

node Node.js

```

1  async lastToJson() {
2      const result = [];
3      const usernameSet = new Set();
4      const last = await this.last(); // "last" 명령어를 통해 받은 정보
5
6      const user = last.trim().split("\n");
7      user.forEach((line) => {
8          const regEx = / 정규식 /;
9          if (regEx.test(line)) {
10              const [, username, terminal, ip, loginTime, logoutTime, connecting] = line.match(regEx);
11              const userInfo = {
12                  "username": username,
13                  "terminal": terminal,
14                  "ip": ip,
15                  "loginTime": loginTime,
16                  "logoutTime": logoutTime ? logoutTime : "",
17                  "connecting": connecting ? true : false
18              };
19              if (!usernameSet.has(username)) {
20                  result.push(userInfo);
21                  usernameSet.add(username);
22              }
23      });
24  });
25  return result;
26 }

```

Set을 통한
중복 제거

중복 제거

- last 명령어는 최근 몇 일간의 접속 기록이 있는 모든 사용자를 표시
- 때문에, 동일한 사용자의 이름이 여러 번 반복
- 이를 방지하기 위해 Set 자료구조를 사용하여 효율적인 중복 제거

command.js

6. Code Analysis

백엔드 코드분석

node Node.js



```

1  class Server {
2      constructor(id) {
3          this.id = id;
4          this.serverList = null;
5          this.reloadTime = 1.5;
6      }
7
8      setServer() {...} // 5개의 Command 클래스 생성
9
10     fileWrite(fileName, content) {...} // JSON파일 쓰기
11
12     async serverWork() {} // 1.5초 간격으로 실행행
13
14     serverRun() {...} // Controller
15 }
16
17 const server = new Server("server");
18 server.run();

```

server.js

Server 클래스

- 필요한 Linux 명령어의 결과를 파싱하는 Commands 클래스에서 데이터를 가져옴
- 이를 주기적으로 실행하고 JSON 파일로 저장할 수 있도록 별도의 클래스를 구성

6. Code Analysis

백엔드 코드분석

node Node.js

EX Express

```

1  class Service {
2      constructor(id) {...} // 속성
3
4      init() {...} // Express 설정
5
6      runServer() { // server.js 하위 프로세스로 실행
7          const path = require("path");
8          const { spawn } = require("child_process");
9
10         const serverProcess = spawn("node", [path.join(__dirname, "..", "server", "server.js")], {
11             stdio: "inherit",
12         });
13     }
14
15     route() { /* 다음 페이지에서 자세히 */ }
16
17     listen() {
18         const { port } = require("./config/daemon.js");
19
20         this.app.listen(port, () => {
21             console.log(`http://localhost:${port}`);
22         });
23     }
24     service() {...} // Controller
25 }
26
27 const service = new Service("service");
28 service.service();

```

app.js

Express

- Express 서버를 실행하기 위한 클래스
- 본 클래스는 Express를 설정하고, 라우팅 및 포트 설정 등을 수행
- server.js를 Express 서버가 실행될 때 자동으로 하위 프로세스로 실행되도록 설정

6. Code Analysis

백엔드 코드분석

node Node.js

EX Express

```

1 route() {
2   const path = require("path");
3   const route_JSON = require(path.join(__dirname, "/route", "/route_JSON.js"));
4   const get_total_page_info = require(path.join(__dirname, "/route", "/get_total_page_info.js"));
5   const get_total_all_info = require(path.join(__dirname, "/route", "/get_total_all_page.js"));
6   const ChatBot = require(path.join(__dirname, "/route", "/chat_bot.js"));
7
8   // CLI command Area
9   this.app.use("/server", route_JSON("server"));
10  this.app.use("/network", route_JSON("ifconfig"));
11  this.app.use("/runtime", route_JSON("runtime"));
12  this.app.use("/status", route_JSON("status"));
13  this.app.use("/ping", route_JSON("ping"));
14  this.app.use("/user_list", route_JSON("userList"));
15  this.app.use("/used_port", route_JSON("usedPort"));
16  this.app.use("/active_port", route_JSON("activePort"));
17
18  ` // Router Explanation
19  /network => ifconfig 값 ifconfig.json
20  /runtime => 서버 실행 시간(uptime) runtime.json
21  /status => memory, cpu, disk 정보 status.json
22  /ping => 네트워크 지연 시간(ping -c 1 8.8.8.8) ping.json
23  /user_list => 현재 접속 중인 유저(last) userList.json
24  /used_port => n번 보트에 m개 접속중(ss) usedPort.json
25  /active_port => 현재 열려있는 모든 포트 확인(netstat) activePort.json
26
27
28  // DB Area
29  this.app.use("/get_total_page_info", get_total_page_info());
30  this.app.use("/get_total_all_info", get_total_all_info());
31
32  // Chat Bot Area
33  const chat_bot = new ChatBot("chat_bot")
34  this.app.use("/chat_bot", chat_bot.run());
35 }

```

app.js

Express - route 메서드

- Router 객체를 분리하여 관리하기 때문에, route() 메서드에서 이를 받음
- 직접한 경로와 Router 객체를 매칭
- 저장된 JSON 파일을 읽어 반환
- DB에서 데이터를 가져와 전달
- 챗봇과 사용자를 연결

6. Code Analysis

백엔드 코드분석

node Node.js

EX Express

Open Ai

```

1 class ChatBot {
2   constructor(id) {...}
3   init() { // Router 객체 호출
4     const express = require("express");
5     this.router = express.Router();
6   }
7   prompt() {
8     return `내가 질문을 하면 최대한 네트워크나 서버에 관련된 질문에만 대답해줘.
9       만약 네트워크나 서버에 관련된 질문이 아니라면 네트워크에 대한 질문을 해주세요. 같은 느낌으로 대답해줘
10      나의 질문은 : `.trim();
11   }
12   async callChatBot(question = "") {
13     require("dotenv").config();
14     const OpenAI = require("openai");
15     const openai = new OpenAI({
16       apiKey: process.env.OPENAI_API_KEY
17     });
18     try {
19       const completion = await openai.chat.completions.create({
20         model: "gpt-4o-mini",
21         messages: [
22           { role: "user", content: `${this.prompt()}${question}` },
23         ],
24       });
25       return completion.choices[0].message;
26     } catch (error) { return error; }
27   }
28   active() {
29     this.router.post("/", async (req, res) => {
30       const { question } = req.body;
31       const answer = await this.callChatBot(question);
32       res.send(answer);
33     });
34   }
35   run() {...} // Controller
36 }
37 module.exports = ChatBot;

```

Open Ai 호출 및
프롬프트와 질문 접합

chat_bot.js

하나의 클래스로 구현한 챗봇 기능

- 챗봇 Router를 하나의 클래스로 구성
- .env 파일을 통한 API 키 관리
- 사용자의 질문을 네트워크 및 서버 관련 내용으로 제한
- 반환된 대답을 Express서버에서 사용 가능
- 최종적으로 Router 객체 반환

Review

후기

7. Review

후기 - 서동현

“시련을 넘어, 우리팀은 성장했다!”

네트워크 대시보드 프로젝트를 막연한 자신감으로 시작했지만, 초기부터 개발환경 통일의 어려움을 겪었습니다. 팀원들이 각자 다른 운영체제인 Windows와 Mac을 사용했고, 실제 서비스 환경은 Ubuntu였기에 Ubuntu와 DB 버전을 통일하고 소스를 호스트 볼륨으로 연결하여 개발환경을 맞추었습니다.

또한, 페이지 기능이나 대시보드에 표시할 특정 데이터에 대해 논의하는 과정에서 서로 같은 이야기를 하고 있다고 생각했지만, 이해 차이로 의견이 엇갈린 경험이 있었습니다. 공공 API를 사용하는 것이 아니라, 우리 팀이 직접 정한 데이터를 API 서버에 담아 운영하기까지 많은 회의 시간이 소요되었습니다. 이러한 경험을 바탕으로 앞으로는 기능을 좀 더 구체적으로 문서화하여 시간과 프로젝트 효율을 높여야겠다는 생각을 하게 되었습니다.

이전 프로젝트와는 달리 효율을 높이기 위해 프론트엔드와 백엔드를 분업하였는데, 처음에는 작업 속도가 느렸지만 점차 각자의 역할을 명확히 이해하며 진행할 수 있어 매우 효율적이었습니다.

저는 DB 설계 및 데이터 저장 역할을 맡아 시스템정보 모듈을 통해 실시간 데이터를 수집하고, 시스템 로그를 기반으로 유사 로그를 파싱하여 데이터베이스에 저장하는 작업을 수행했습니다. 이를 위해 코드를 모듈별 클래스로 작성하고, 하나의 클래스에서 모든 작업이 진행되도록 구성하였습니다. 이 과정에서 객체지향 프로그래밍에 대해 더욱 깊이 이해할 수 있었습니다.

비록 쉽지 않은 프로젝트였지만, 우리 팀의 협업 능력이 발휘되어 기한 내에 무사히 프로젝트를 마칠 수 있었습니다. 이번 프로젝트를 통해 서버와 네트워크에 대한 지식뿐 아니라, 협업, 개발환경 통일의 중요성을 깊이 깨닫게 되었습니다.

7. Review

후기 - 배주환

“값 처리 하는자” “파싱의 왕” “태양의 코드” “나”

네트워크 대시보드라는 주제가 정해졌을 때, 네트워크에 대한 지식이 부족했던 저는 어떤 정보를 어떤 형태로 표시해야 할지 고민하며, 팀원들과 함께 구상하고 회의를 거듭하며 프로젝트를 진행해 나갔습니다.

표시할 데이터와 사용할 명령어가 정리된 후에는, 프론트엔드와 백엔드로 역할을 나누고 서로 주고받을 JSON 데이터의 형태를 합의했습니다. 백엔드를 담당한 저는 이전 프로젝트와 달리 외부 API를 호출하는 것이 아니라 직접 API를 만들어야 했습니다.

정의한 값들이 정확하게 구현되지 않거나 형태가 달라지면 프론트엔드의 작업이 어려워지기에, 빠르게 코드를 작성하는 것보다 먼저 깊이 고민하는 시간을 가졌습니다. 신중하게 검토하고 검증한 후 코딩을 시작했기 때문에, 보다 구조적이고 의미 있는 코드를 작성하는 능력을 기를 수 있었습니다.

Express를 이용해 백엔드의 라우팅 작업을 마친 뒤에는 프론트엔드 작업을 도와 데이터를 화면에 표시하는 역할도 맡았습니다.

중간에 투입되었음에도, 팀원들이 섬세하게 컴포넌트를 나누어 구성한 덕분에 빠르게 적응하고 효율적으로 작업을 진행할 수 있었습니다.

또한, 백엔드와 프론트엔드의 중간다리 역할을 하며 양측의 의견을 조율하고, 새로운 데이터를 추가하거나 기능을 확장하는 과정에서 적극적으로 구현을 진행했습니다.

이번 프로젝트를 성공적으로 마치면서, 네트워크와 서버에 대한 이해뿐만 아니라 API 설계와 데이터 구조에 대해 깊이 생각할 수 있는 시간을 가졌습니다. 또한, 팀원들과의 협업을 통해 완벽한 분업과 효과적인 커뮤니케이션 능력을 더욱 향상시킬 수 있었습니다.

7. Review

후기 - 서현지

“프론트엔드 도전! Type Script, D3.js, 그리고 네트워크 대시보드”

우리 팀은 이번 프로젝트에서 네트워크 대시보드를 개발했습니다. 저는 프론트엔드 담당으로서 주로 React와 TypeScript, Sass를 사용해 사용자 인터페이스를 구현했고, D3.js를 이용해 네트워크 데이터(트래픽, 패킷, 연결 상태 등)를 쉽게 볼 수 있는 그래프를 만들었습니다.

프로젝트 진행 중에는 API로부터 데이터를 받아오고, 그 데이터를 차트로 시각화하는 작업이 많았습니다. 처음 사용해 보는 D3.js라서 처음엔 어려움이 많았지만 하나하나 구현해 가면서 D3를 자연스레 능숙하게 사용하게 되었습니다. 특히 데이터를 동적으로 업데이트하고 사용자에게 직관적인 정보를 제공하는 데 집중했습니다. 이런 작업 덕분에 복잡한 네트워크 상태를 한눈에 파악할 수 있도록 만들 수 있었습니다.

또한, 예전에 취득한 네트워크 자격증이 이번 프로젝트에서 큰 도움이 되었습니다. 네트워크의 기본 원리와 실무 적용 사례에 대한 이해를 바탕으로 대시보드의 데이터 시각화와 상태 관리 로직을 구현할 수 있었고, 이를 통해 보다 효율적이고 정확한 데이터 표현이 가능해졌습니다. 이 경험은 단순히 기술적인 측면뿐만 아니라, 실제 네트워크 환경에서 발생할 수 있는 다양한 문제들을 예측하고 해결하는 데도 긍정적인 영향을 주었습니다.

더불어, Docker를 사용하여 개발 환경을 세팅했기 때문에 모든 팀원이 동일한 환경에서 작업할 수 있었고, 이에 따라 환경 차이로 인한 충돌이나 에러 없이 개발을 수월하게 진행할 수 있었습니다.

팀원 모두가 각자의 역할에 최선을 다하며 꾸준하게 소통해 준 덕분에, 단순한 기능 구현을 넘어서 실제 현장에서 사용할 수 있는 수준의 네트워크 모니터링 도구를 만들어낼 수 있었던 것 같습니다. 모두의 노력과 협업 덕분에 이번 프로젝트가 성공적으로 마무리된 것에 대해 진심으로 감사하게 생각합니다!

7. Review

후기 - 윤혜경

“함께했기에 배울 수 있었던 것들”

프로젝트 주제를 처음 접한 후, 우리 팀은 수시로 모여 회의를 진행했습니다. 다른 팀에 비해 훨씬 더 많은 시간을 회의에 할애했지만, 긴 회의 시간 동안 단 한 번도 힘들게 느껴지지 않았습니다. 오히려 나와는 다른 다양한 시각을 인정하고 배우며 함께 고민할 수 있었던 소중한 시간으로 기억됩니다. 또한, 팀원 개개인이 어떤 장점과 능력을 가지고 있는지 파악할 수 있었던 값진 경험이기도 했습니다.

이같이 긴 회의 끝에 우리는 **흔들림 없이 단단한 목표**를 세울 수 있었습니다. 프로젝트가 진행되는 동안, 우리는 모두 같은 방향을 향해 나아갔습니다. 충분히 긴 회의를 통해 모든 요구사항과 세부사항을 명확하게 조율했기 때문이라고 생각합니다. 이어진 개발 과정에서도 큰 혼선 없이 계획대로 원활하게 진행될 수 있었습니다. 팀원 각자가 잘할 수 있는 부분을 찾아 각자의 역할을 분명히 나누었으며, 단 한 사람도 예외 없이 맡은 바를 책임감 있게 해냈기에 빠르고 정확한 작업이 가능했습니다.

이번 프로젝트는 제가 경험한 첫 팀 프로젝트였습니다. 같은 문제를 서로 다른 관점에서 바라볼 수 있다는 점에 놀라기도 했고, 서로 다른 작업 환경에서 하나의 작업을 수행하는 과정에서 나타나는 문제들에 당황하기도 했습니다. 처음부터 끝까지 혼자서 작업했던 개인 프로젝트에서는 결코 얻을 수 없었던 귀중한 경험이었습니다. 프로젝트를 혼자 진행할 때에는 계획의 중요성을 크게 체감하지 못했었는데, 팀 프로젝트를 통해 **단단히 정의한 명세들이 엄청난 힘을 발휘한다는 것을 배울 수 있었습니다.**

앞으로 이 분야에서 일을 하게 된다면, 수없이 많은 팀 프로젝트를 경험하게 될 것입니다. 함께 의견을 나누고 서로의 생각을 존중하며 가장 좋은 답안을 찾아가던 이번 경험이 좋은 밑거름이 될 것이라 확신합니다. 서로에게 힘이 되어주며 마지막까지 손잡고 달려온 팀원들에게 깊은 고마움을 전하고 싶습니다.



Thank you