

Team

Nody Chat

김시준, 김신명, 신현식, 이민서, 정은혜

팀원 소개

김시준
(팀장)

socket 통신 구현
로직설계
DB 설계
백엔드 작업
프론트js 작업
API작업 총괄
(sms, youtube, ip,
weather 등)

김신명

PPT
DB ERD
채팅방 백엔드 작업
봇 차단코드 작성

신현식

API 명세서
Figma
socket 통신
실시간 채팅 구현 작업

이민서

뉴스 기능 크롤링 작업
TTS 기능 작업
reCaptcha api 작업
친구 요청 및 삭제

정은혜

디자인 구성
Figma
HTML/CSS
반응형 웹 작업

Contents

01. 프로젝트 개요

02. 개발 목표

03. 기술 스택

04. 시스템 아키텍처

05. 설계

06. API 명세서

07. Figma

08. QA Sheet

09. 트러블 슈팅

10. 기능 시연

1. 프로젝트 개요

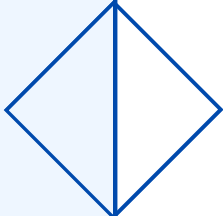
이름 : WEBCHATTING

설명 : JWT 인증 기반의 실시간 채팅 웹사이트

핵심 기능

- 회원가입 / 로그인
- 친구추가 / 채팅방 생성, 초대, 나가기
- 채팅방별 실시간 메시지 전송 및 저장
- 명령어 봇 기능

2. 개발 목표

- 
- 사용자 인증 및 실시간 통신 학습
 - RESTful API와 실시간 Socket 병행 처리
 - 웹 크롤링을 통한 원하는 데이터 파싱

3. 기술 스택

영역	기술
Backend	Node.js, Express
Database	MongoDB Atlas, Mongoose
인증	JWT, bcrypt, reCaptcha
실시간 통신	Socket.io
프론트	HTML, CSS, JavaScript
API 명세서	Notion

4. DB - 구성

```
const userSchema = new mongoose.Schema(  
  {  
    nickname: { type: String, required: true, unique: true },  
    email: { type: String, required: true, unique: true },  
    phone: { type: String, required: true, unique: true },  
    userid: { type: String, required: true, unique: true },  
    passwd: { type: String, required: true },  
    friend: [{ type: String }],  
    createdAt: { type: Date, default: Date.now },  
  },  
  { versionKey: false }  
);
```

users 컬렉션
구성 사항

4. DB - 구성

```
const roomSchema = new Mongoose.Schema(  
  {  
    name: { type: String, required: true },  
    members: {  
      type: [{ type: String, required: true }],  
      required: true,  
    }, // 참가자 userid 배열로 저장  
    createdAt: { type: Date, default: Date.now },  
  },  
  { versionKey: false }  
);
```

rooms 컬렉션
구성 사항

4. DB - 구성


```
const messgeSchema = new mongoose.Schema(  
  {  
    chatroom: { type: String, required: true },  
    sender: { type: String, required: true },  
    content: { type: String, required: true },  
    createdAt: { type: Date, default: Date.now },  
  },  
  { versionKey: false }  
);
```

messges 컬렉션
구성 사항

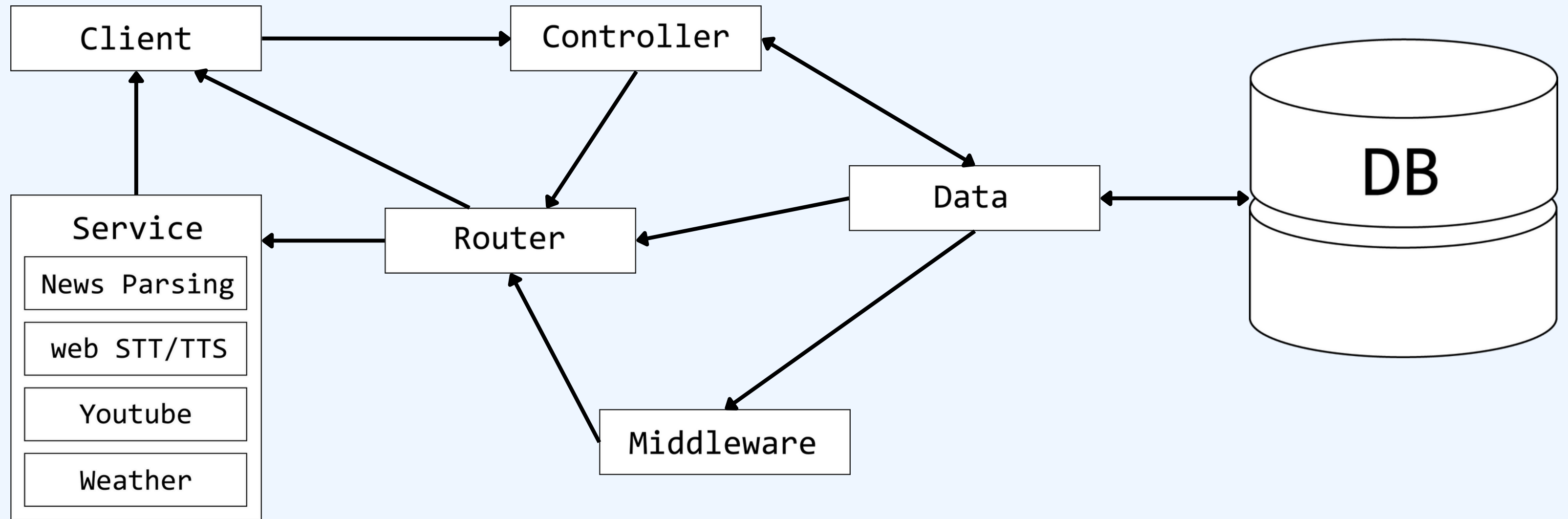
4. DB - ERD

users 사용자 정보					
	_id	VARCHAR	N-N	UQ	default
	nickname	VARCHAR	N-N	UQ	default
	email	VARCHAR	N-N	UQ	default
	userid	VARCHAR	N-N	UQ	default
	passwd	VARCHAR	N-N	UQ	default
	image	VARCHAR	NULL	UQ	default
	friend	VARCHAR[]	NULL	UQ	default
	createdAt	DATETIME	NULL	UQ	Date.now

chatrooms 채팅방					
	_id	ObjectId	N-N	UQ	default
	name	VARCHAR	N-N	UQ	default
	members	VARCHAR[]	N-N	UQ	default
	createdAt	DATETIME	NULL	UQ	Date.now

messages 메시지					
	_id	ObjectId	N-N	UQ	default
	chatroom	VARCHAR	N-N	UQ	default
	sender	VARCHAR	N-N	UQ	default
	content	TEXT	N-N	UQ	default
	createdAt	DATETIME	NULL	UQ	Date.now

5. 설계 - 시스템 아키텍처



5. 설계 - 보안코드

```
const limiter = rateLimit({
  windowMs: 60 * 1000,
  max: 30,
  message: "요청이 너무 많습니다. 잠시 후 다시 시도해주세요.",
  standardHeaders: true,
  legacyHeaders: false,
})
```

시간내 api 호출 횟수를 제한하는 코드

- 분당 채팅 포함전체 서비스 기준으로 500회, 문자 서비스 기준으로 10회 제한
실제 서비스 규모에 따라 조정 필요

5. 설계 - 보안코드



```
app.get("/robots.txt", (req, res) => {  
  res.type("text/plain");  
  res.send("User-agent: *\nDisallow: /");  
});
```

Get /robots.txt

- (url)/robots.txt 요청시
텍스트로 robots.txt 내용 출력



robots.txt

```
User-agent: *  
Disallow: /
```

Disallow: /

- 모든 User-agent에 대해 접속을 금지

5. 설계 - 보안코드

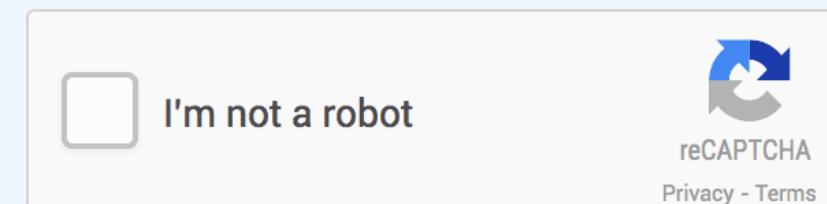
```
const botUserAgents = [
  /googlebot/i, /bingbot/i, /slurp/i, /duckduckbot/i,
  /baiduspider/i, /yandexbot/i, /sogou/i, /exabot/i,
  /facebot/i, /ia_archiver/i, /python-requests/i,
  /scrapy/i, /curl/i, /wget/i,
];
app.use((req, res, next) => {
  const ua = req.headers["user-agent"];
  if (ua && botUserAgents.some((bot) => bot.test(ua))) {
    return res.status(403).send("Forbidden");
  }
  next();
});
```

지정된 검색엔진 봇과
headers 에 “user-agent”가 있으면
403 : Forbidden 반환

5. 설계 - 보안 API

```
export async function verifyRecaptcha(req, res, next) {
  try {
    const recaptchaToken = req.body["g-recaptcha-response"];
    if (!recaptchaToken) {
      return res
        .status(400)
        .send(
          `<script>alert('reCAPTCHA 토큰이 없습니다.');
```

reCaptcha 적용



5. 설계 - API

```
var uri = `https://api.coolsms.co.kr/messages/v4/list?limit=1`;
axios
  .get(uri, {
    headers: {
      Authorization: `HMAC-SHA256 apiKey=${apiKey}, date=${now},
salt=${salt}, signature=${signature}`,
    },
  })
  .then((res) => {
    console.log(res.data);
  })
  .catch((error) => {
    console.log(error.response.data);
  });
```

CoolSMS

- 회원가입 및 비밀번호 재설정을 위한 인증번호를 문자 메시지를 보내고 확인하기 위한 API

5. 설계 - API

```
const recognition = new window.webkitSpeechRecognition();
recognition.lang = "ko-KR";
recognition.interimResults = false;
recognition.continuous = false;

recognition.onresult = (event) => {
  const transcript = event.results[0][0].transcript;
  console.log("🗣️ 텍스트:", transcript);
  sendToChat(transcript);
};

recognition.onerror = (e) => {
  console.error("STT 오류:", e.error);
};

recognition.start();
```

Web STT (Speech To Text)

- 브라우저에서 음성을 텍스트로 바꾸고 출력하는데 필요한 API

5. 설계 - API

```
function playTTS(message) {  
  if (!isTTSEnabled) return;  
  if (!window.speechSynthesis) {  
    console.warn("이 브라우저는 TTS를 지원하지 않습니다.");  
    return;  
  }  
  window.speechSynthesis.cancel();  
  const utterance = new SpeechSynthesisUtterance(message);  
  utterance.lang = "ko-KR";  
  utterance.pitch = 1;  
  utterance.rate = 1;  
  utterance.volume = 1;  
  if (koreanVoice) utterance.voice = koreanVoice;  
  window.speechSynthesis.speak(utterance);  
} });
```

Web TTS (Text to Speech)

- 브라우저에서 텍스트를 음성으로 바꾸고 출력하는데 필요한 API

5. 설계 - API

```
router.get("/", async (req, res) => {
  const { query } = req.query;
  const apiKey = config.youtue.apiKey;
  if (!query) {
    return res.status(400).json({ error: "검색어가 없습니다." });
  }
  const apiUrl = `https://www.googleapis.com/youtube/v3/search?
part=snippet&type=video&maxResults=1&q=${encodeURIComponent(
  query
)}&key=${apiKey}`;
  try {
    const response = await fetch(apiUrl);
    const result = await response.json();

    const videoId = result.items?.[0]?.id?.videoId;
    if (!videoId) {
      return res.status(404).json({ error: "영상이 없습니다." });
    }
    res.json({ videoId });
  } catch (err) {
    console.error("유튜브 API 에러:", err);
    res.status(500).json({ error: "유튜브 검색 실패" });
  }
});
```

Youtube Data API v3

- youtube영상을 탐색하여 출력하기 위한
구글에서 제공하는 유튜브 탐색 API

5. 설계 - API

```
router.get("/", async (req, res) => {
  try {
    const rawIP = req.headers["x-forwarded-for"] ||
req.socket.remoteAddress;
    const clientIP = Array.isArray(rawIP)
      ? rawIP[0]
      : rawIP.split(",")[0].trim();
    console.log("Client IP:", clientIP);
    const geoRes = await axios.get(`http://ip-api.com/json/${clientIP}`);
    if (geoRes.data.status !== "success") {
      return res.status(400).json({ error: "IP 위치 조회 실패" });
    }
    const { lat, lon } = geoRes.data;
    const apikey = config.weather.apiKey;
    const weatherRes = await axios.get(
      `https://api.openweathermap.org/data/2.5/weather?
lat=${lat}&lon=${lon}&lang=kr&units=metric&appid=${apikey}`);
    const { temp, feels_like, temp_min, temp_max, pressure, humidity, sea_level,
      grnd_level, } = weatherRes.data.main;
    res.json({ temp, feels_like, temp_min, temp_max, pressure, humidity, sea_level,
      grnd_level, });
  }
});
```

ip-api

- 클라이언트의 ip를 기준으로
위치 정보(위도, 경도)를 가져오는 API

OpenWeatherMap

- 위치정보에 따른 실시간 날씨정보 데이터를
제공하는 API

5. 설계 - Parsing

```
router.get("/", async (req, res) => {
  try {
    const url = "https://media.naver.com/press/056";
    const response = await axios.get(url, {
      headers: {
        "User-Agent":
          "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36",
      },
    });
    const $ = cheerio.load(response.data);
    const newsItems = [];
    $(".ofra_list li.ofra_list_item.as_thumb").each((i, el) => {
      const num = $(el).find("i.ofra_list_num").text().trim();
      const headline = $(el).find(".ofra_list_tx_headline").text().trim();
      const date = $(el).find(".ofra_list_tx_date").text().trim();
      const views = $(el).find(".ofra_list_tx_visit").text().trim();
      const url = $(el).find("a").attr("href");
      const img = $(el).find(".ofra_list_img img").attr("src") || null;
      newsItems.push({ num, headline, date, views, url, img });
    });
  }
});
```

백엔드에서 뉴스를 파싱하는 js

- 뉴스 페이지 크롤링을 통한 실시간 순위에 따른 뉴스 정보들을 파싱해 해와서 그 값들을 프론트로 넘겨줌

axios, cheerio 모듈 사용

- **axios** : HTTP 요청을 보내기 위한 모듈
get, post 등으로 HTML 코드나 JSON 데이터를 받아와서 출력 하기 위해 사용
- **cheerio** : HTML 파싱 및 DOM 탐색을 위한 모듈
웹 페이지의 HTML 코드를 가져와서 원하는 데이터를 추출하기 위해 사용

5. 설계 - Socket

```
robots.txt

io.use(async (socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) {return next(new Error("토큰 없음"));}
  try {
    const secret = config.jwt.secretKey || "your_jwt_secret";
    const decoded = jwt.verify(token, secret);
    const IdFromToken = decoded.id || decoded._id;
    const user = await userRepository.findById(IdFromToken);
    if (!user) {return next(new Error("사용자 없음"));}
    console.log(user);
    socket.userId = user.userid;
    socket._mongoId = user._id;
    socket.friend = user.friend;
    next();
  } catch (err) {
    console.error("JWT 인증 실패:", err.message);
    next(new Error("인증 실패"));
  }
});
```

Socket.io

- 실시간 변경사항을 화면에 업데이트하기 위해서 사용
- 실시간 채팅, 채팅방 생성, 유저 초대 등에 이용됨

6. API 명세서

</> AI활용 웹 개발 세미프로젝트 API 명세서(Nody Chat)

All Documentation	By Section	Endpoints List	+							새로 만들기	
Router	Title	Section	Method	Endpoint	설명	인증 여부	Request Body	Response	Status Code	추가 설명	+
API Overview	API 개요	Overview	N/A	N/A	WebChatting API						
User	</> 로그인	Endpoints	POST	/user/login	사용자가 아이디와 비밀번호로 로그인합니다.	✓	{ "userid": "apple", "password": "123456" }	200: OK 토큰 인증 성공시	200: OK 404: Not Found 400: Bad Request 500: Server Error	인증에 성공하면 JWT 토큰이 쿠키에 저장되고, /main 페이지로 리디렉션됩니다.	
	</> 유저 아이디와 닉네임 매핑	Endpoints	GET	/user/user-map	아이디와 닉네임을 매핑 시켜데이터베이스에서 정보를 가져오기 위한 기능		{ "nicknames": ["user.nickname"] }	200: OK { "userid": "nickname" }	500: Server Error 200: OK	페이지에선 닉네임으로 사용되기 때문에 닉네임과 아이디를 매핑시켜 데이터베이스에 아이디로 정보를 요청하고 저장	
	</> 닉네임과 유저 아이디 매핑	Endpoints	POST	/user/nicknames-to-ids	닉네임과 아이디를 매핑시켜 아이디 값에 맞는 정보를 데이터베이스에 보내기 위한 기능		{ "friend": ["user.nickname"] }	200: OK { "userids": ["user.nickname"] }	200: OK 400: Bad Request 500: Server Error	반대로 데이터를 데이터베이스에 보내기 위해서 페이지의 닉네임과 매핑되는 아이디 값을 받아 데이터베이스와 연결	
	</> 로그인한 사용자의 친구추가	Endpoints	POST	/user/add-friend	친구 추가 기능을 제공합니다.	✓	{ "newNickname": "newNickname" }	200: OK { "message": "newNickname" }	200: OK 400: Bad Request 404: Not Found 409: Conflict	친구 검색 란에 닉네임을 입력하면 같은 입력 값의 유저가 출력 되고 추가 버튼 클릭 시 친구 추가가 되고 친구 목록에 추가 합니다.	
	</> 로그인한 사용자 개인정보 보내주기	Endpoints	GET	/user/me	마이페이지에 들어갈 회원 정보를 데이터베이스에서 가져옵니다.	✓	{ "userid": "apple" }	200: OK { "userid": "apple" }	200: OK 404: Not Found		
	</> 로그인한 사용자의 친구 목록 보내주기	Endpoints	GET	/user/get-friend	유저 아이디에 저장된 친구 목록을 데이터베이스에서 가져옵니다.	✓	{ "userid": "apple" }	200: OK { "friend": ["banana"] }	200: OK 404: Not Found		
	</> 로그인한 사용자의 친구 삭제	Endpoints		/user/remove-friend	친구 목록에 있는 유저를 삭제하는 기능	✓	{ "nickname": "멜론" }	200: OK { "message": "멜론" }	200: OK 400: Bad Request 404: Not Found		
	</> 회원가입	Endpoints	POST	/user/signup	새 사용자를 생성합니다.	✓	{ "nickname": "김사과", "password": "123456", "email": "apple@apple.com", "phone": "01011111111" }	201: created { "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1d2UiOiJkaWUyIiwiaWF0IjoxNjE1MjM0MjE5LCJ1d2UiOiJkaWUyIn0" }	201: Created 400: Bad Request 409: Conflict 500: Server Error	이메일, 전화번호, 닉네임, 아이디는 중복될 수 없습니다.	
	</> 유저 중복 확인 버튼	Endpoints	POST	/user/check	회원가입 시 필수 입력값이 같은 유저가 있는지 확인합니다.		{ "nickname": "김사과", "password": "123456" }	409: Conflict { "message": "중복된 유저입니다." }	200: OK 409: Conflict		
	</> 아이디 찾기	Endpoints	POST	/user/findID	회원가입 시 입력했던 이메일로 아이디를 찾습니다.		{ "email": "apple@apple.com" }	200: OK { "userid": "apple" }	200: OK 401: Unauthorized		
	</> 회원가입용 인증번호 보내기	Endpoints	POST	/user/send-signupCode	회원가입 시 유저가 입력한 전화번호로 문자 인증 번호를 보내는 기능을 합니다.		{ "phone": "01011111111" }	{ "message": "인증번호가 전송되었습니다." }	200: OK 500: Server Error	외부 API인 coolsms를 활용했습니다. 인증 번호의 유효 시간은 3분으로 설정, uuid로 생성한 세션 키를 폰 번호와 매칭시켜 이후 인증 시 세션 키를 활용	

7. Figma

8. QA Sheet

QA 시트

☆

🔗

📁

파일

수정

보기

삽입

서식

데이터

도구

확장 프로그램

도움말

🔍 메뉴

↶

↷

🖨

🔊

100%

W

%

0.00

123

기본값 ...

10

+

B

I

↶

A

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

📄

🔍

9. 트러블 슈팅

1. 날씨 명령어봇의 문제점

초기에 날씨 명령어 봇의 구성은 네이버 날씨 검색 html 페이지를 백엔드에서 axios, cheerio 모듈을 사용하여 파싱하여 필요한 데이터를 프론트에 넘겨주는 방식으로 제작됨 ➡ 로컬 서버에서는 잘 돌아가나 클라우드 타입 배포시 네이버 검색 페이지 측에서 봇으로 인식하여 차단되는 현상이 발생됨.

문제 해결: 초기의 파싱으로 날씨 데이터를 넘겨주던 로직을 ip-API와 OpenWeatherMap API 를 사용한 날씨 API 이용 로직으로 교체함

2. 뉴스 명령어봇의 문제점

뉴스 검색결과 출력은 백엔드에서 axios, puppeteer(크롬 브라우저를 자동으로 제어할 수 있게 해주는 모듈) 모듈을 이용하여 결과를 파싱하여 프론트에 넘겨주는 방식으로 제작됨. ➡ 로컬 서버에서는 잘 돌아가나 클라우드 타입 배포시 백엔드 서버측에서 크롬브라우저 지원이 안되어 뉴스 파싱이 되는 현상이 발생함.

문제 해결: 크롤링 방식을 axios와 cheerio 조합으로 변경

9. 트러블 슈팅

3. 새로운 채팅방 생성시 상대방 유저에게 실시간으로 채팅방에 초대된 것이 보이지 않던 문제점

소켓 통신을 통한 실시간 채팅은 지원하나, 새로운 채팅방 생성시 이미 접속중인 초대된 상대방에게 페이지를 새로고침 하기 전까지 보여지지 않는 문제점.

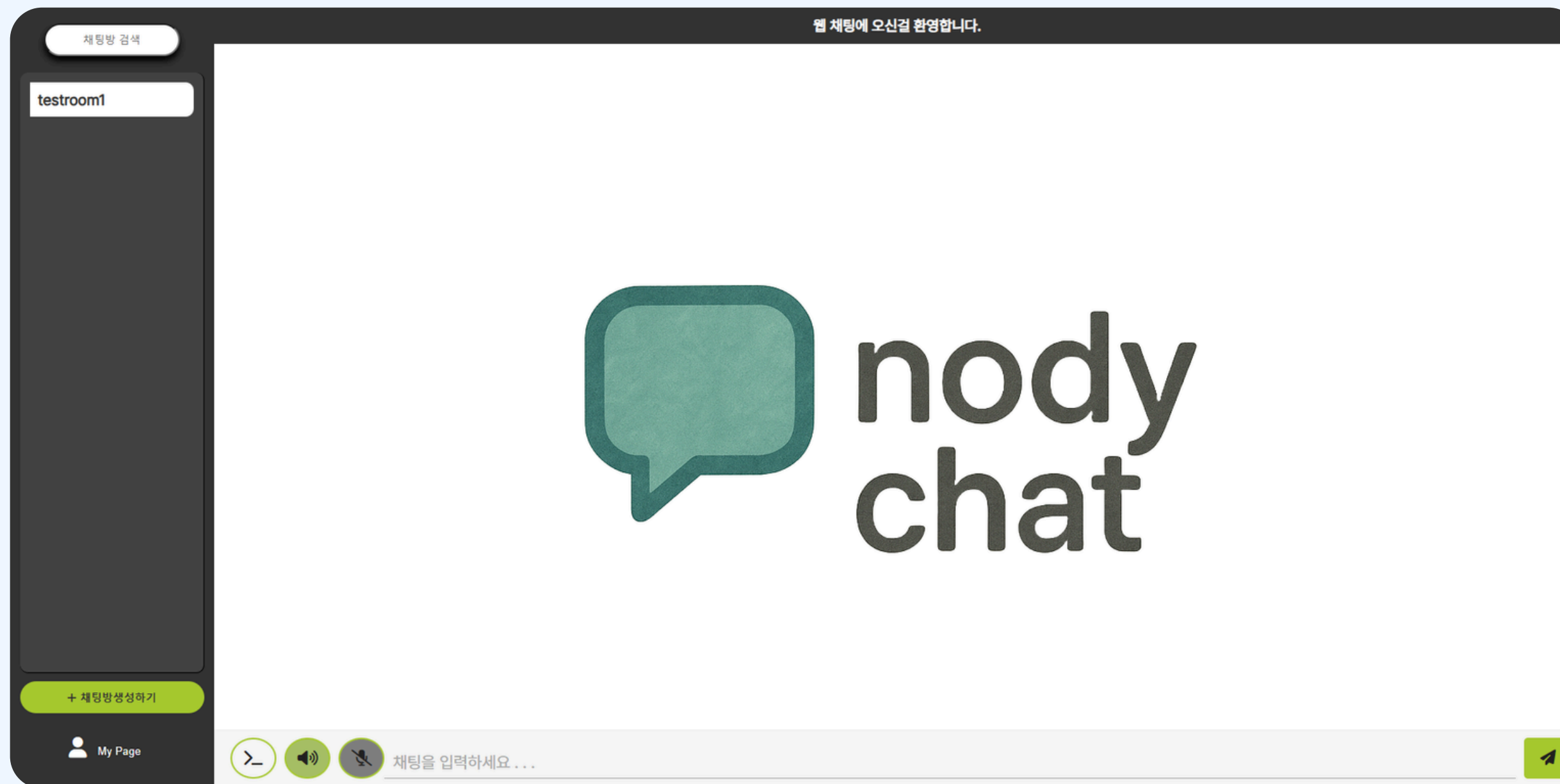
문제 해결: 채팅 뿐만이 아니라 메인페이지 로딩시에 소켓에 연결을 하고, 백엔드에서 채팅방 생성되는 이벤트 작동시 소켓에 접속중인 모든 사용자에게 해당 이벤트를 브로드 캐스팅 하도록 수정함.

4. 채팅방 진입 할 때와 상대방이나 사용자 본인이 채팅을 보냈을 때 스크롤이 맨 아래로 자동 이동하지 않는 문제점

채팅 내용이 길어질 시 스크롤 하단 고정 이벤트가 등록되어있음에도 스크롤이 자동으로 자동으로 최근 채팅으로 내려가지 않는 문제점이 발생됨

문제 해결: 스크롤 하단 고정을 함수로 재구성하여 새로운 메시지 생성시에 짧은 간격으로 반복하도록 로직을 수정함.

10. 기능 시연



감사합니다
