

최종 문서

1. 호스팅 및 서버 설정

```
require('dotenv').config({ path: require('path').join(__dirname, '.env') }); //
환경 변수 로드. backend/.env를 명시적으로 지정

const mongoose = require('mongoose');
const https = require('https');
const http = require('http');
const fs = require('fs');
const path = require('path');
const selfsigned = require('selfsigned');
const app = require('./app'); // 분리된 app.js를 가져옵니다.

// 포트와 환경 설정 (하드코딩)
const HTTPS_PORT = 3000; // 내부 HTTPS 포트 (고정)
const HTTP_PORT = 80; // HTTP 표준 포트 (외부 접속용)
const NODE_ENV = 'production'; // 프로덕션 모드 (고정)
const MONGODB_URI = process.env.MONGODB_URI || 'mongodb://yes231
0.duckdns.org:27017/scheduleApp';

// DB 연결
mongoose.connect(MONGODB_URI)
  .then(() => console.log('✅ MongoDB 연결 성공'))
  .catch(err => console.error('❌ MongoDB 연결 실패:', err));

// HTTP에서 HTTPS로 리디렉션하는 미들웨어
const redirectToHTTPS = (req, res) => {
  const host = req.headers.host.split(':')[0]; // 포트 번호 제거
  const redirectUrl = `https://${host}`;
  console.log('🔄 HTTP → HTTPS 리다이렉트: ${req.url} → ${redirectUrl}${r
eq.url}`);
  res.writeHead(301, { Location: `${redirectUrl}${req.url}` });
  res.end();
};
```

```

// SSL 인증서 경로 (Cloudflare Origin Certificate)
const sslKeyPath = path.join(__dirname, 'ssl', 'cloudflare-key.key');
const sslCertPath = path.join(__dirname, 'ssl', 'cloudflare-cert.pem');

let sslOptions;

// SSL 인증서 확인 및 로드
if (fs.existsSync(sslKeyPath) && fs.existsSync(sslCertPath)) {
  sslOptions = {
    key: fs.readFileSync(sslKeyPath),
    cert: fs.readFileSync(sslCertPath),
  };
  console.log('🔒 Cloudflare Origin Certificate 사용 중');
} else {
  // SSL 인증서가 없으면 자체 서명 인증서 생성
  console.log('⚠️ SSL 인증서를 찾을 수 없습니다. 자체 서명 인증서를 생성합니다...');

  const attrs = [{ name: 'commonName', value: 'localhost' }];
  const pems = selfsigned.generate(attrs, { days: 365 });

  sslOptions = {
    key: pems.private,
    cert: pems.cert,
  };
  console.log('🔑 자체 서명 SSL 인증서 생성 완료');
}

// HTTP 리디렉션 서버 생성 및 시작
const httpServer = http.createServer(redirectToHTTPS);
httpServer.listen(HTTP_PORT, '0.0.0.0', () => {
  console.log('🔄 HTTP 리디렉션 서버 실행 중: <http://0.0.0.0>:${HTTP_PORT} → <https://yes2310.xyz>');
});

// HTTPS 서버 생성 및 시작
const httpsServer = https.createServer(sslOptions, app);
httpsServer.listen(HTTPS_PORT, '0.0.0.0', () => {

```

```
console.log(`🚀 HTTPS 서버 실행 중: <https://0.0.0.0>:${HTTPS_PORT}`);
console.log(`📱 로컬 접속: <https://localhost>:${HTTPS_PORT}`);
console.log(`🌐 외부 접속은 리버스 프록시를 통해: <https://yes2310.xyz>`);
console.log(`💡 리버스 프록시 설정 필요: 80/443 → ${HTTPS_PORT}`);
});
```

설명: HTTPS 서버 설정과 SSL 인증서 관리, MongoDB 연결, HTTP에서 HTTPS로 리디렉션 기능이 구현되어 있습니다.

2. 로그인 기능

백엔드 - 로그인 API

```
// 로그인
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;

    // 사용자 검색
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ error: '이메일 또는 비밀번호가 올바르지 않습니다.' });
    }

    // 비밀번호 확인
    const isValid = await user.comparePassword(password);
    if (!isValid) {
      return res.status(401).json({ error: '이메일 또는 비밀번호가 올바르지 않습니다.' });
    }

    // JWT 토큰 생성
    const token = jwt.sign(
      {
        userId: user._id,
        email: user.email,
        name: user.name
      },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );
    res.json({ token });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: '서버 오류' });
  }
});
```

```

    },
    JWT_SECRET,
    { expiresIn: '24h' } // 토큰 유효 기간: 24시간
  );

  res.json({
    message: '로그인 성공',
    token,
    userId: user._id,
    name: user.name,
    email: user.email
  });

} catch (error) {
  console.error('로그인 에러:', error);
  res.status(500).json({ error: '로그인 처리 중 오류가 발생했습니다.' });
}
});

```

프론트엔드 - 로그인 컴포넌트

```

const handleSubmit = async (e) => {
  e.preventDefault();
  setIsLoading(true);
  setError('');

  try {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password }),
    });

    const data = await response.json();

    if (!response.ok) {

```

```

    throw new Error(data.error || '로그인에 실패했습니다.');
```

```

  }

  // Store token in localStorage
  localStorage.setItem('token', data.token);

  // Call the onLogin callback with user data
  onLogin(data);

} catch (error) {
  setError(error.message);
} finally {
  setIsLoading(false);
}
};

return (
  <div className="min-h-screen flex items-center justify-center bg-gray-100 p-6">
    <div className="bg-white rounded-xl shadow-xl w-full max-w-md p-8">
      <h2 className="text-3xl font-bold text-center text-gray-800 mb-6">
Voice Manager</h2>
      <p className="text-gray-600 text-center mb-8">AI-powered 스케줄러에 로그인하세요</p>

      {error && (
        <div className="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded mb-4">
          {error}
        </div>
      )}

      <form onSubmit={handleSubmit} className="space-y-6">
        <div>
          <label htmlFor="email" className="block text-gray-700 mb-1">이메일</label>
          <input

```

```

        id="email"
        type="email"
        value={email}
        onChange={(e) ⇒ setEmail(e.target.value)}
        required
        className="w-full px-3 py-2 border rounded-md focus:outline-no
ne focus:ring-2 focus:ring-indigo-500"
        placeholder="your-email@example.com"
      />
    </div>

    <div>
      <label htmlFor="password" className="block text-gray-700 mb-
1">비밀번호</label>
      <input
        id="password"
        type="password"
        value={password}
        onChange={(e) ⇒ setPassword(e.target.value)}
        required
        className="w-full px-3 py-2 border rounded-md focus:outline-no
ne focus:ring-2 focus:ring-indigo-500"
        placeholder="••••••••"
      />
    </div>

    <button
      type="submit"
      disabled={isLoading}
      className={`w-full py-3 ${isLoading ? 'bg-indigo-400' : 'bg-indigo
-600 hover:bg-indigo-700'} text-white rounded-md transition`}
    >
      {isLoading ? '로그인 중...' : '로그인'}
    </button>
  </form>
</div>
</div>
);

```

설명: 이메일/비밀번호 기반 로그인, JWT 토큰 생성 및 저장, 로그인 상태 관리가 구현되어 있습니다.

3. 사용자 인증/권한 기능

백엔드 - 인증 미들웨어

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = 'your_jwt_secret_key'; // 실제로는 환경변수로 관리

module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: '인증 토큰이 필요합니다.' });
  }

  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded; // { userId, email, name }
    next();
  } catch (err) {
    return res.status(401).json({ error: '유효하지 않은 토큰입니다.' });
  }
};
```

프론트엔드 - 인증 컨텍스트

```
import React, { createContext, useState, useEffect, useContext } from 'react';
import { jwtDecode } from 'jwt-decode';

const AuthContext = createContext();

export function useAuth() {
  return useContext(AuthContext);
}
```

```

export function AuthProvider({ children }) {
  const [currentUser, setCurrentUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    if (token) {
      try {
        const decoded = jwtDecode(token);
        const currentTime = Date.now() / 1000;

        if (decoded.exp < currentTime) {
          // Token has expired
          logout();
        } else {
          setCurrentUser(decoded);
        }
      } catch (error) {
        console.error('토큰 디코딩 오류:', error);
        logout();
      }
    }
    setLoading(false);
  }, [token]);

  const login = (userData) => {
    localStorage.setItem('token', userData.token);
    setToken(userData.token);
    setCurrentUser(jwtDecode(userData.token));
  };

  const logout = () => {
    localStorage.removeItem('token');
    setToken(null);
    setCurrentUser(null);
  };

```



```

const value = {
  currentUser,
  token,
  login,
  logout,
  isAuthenticated: !!currentUser,
};

return (
  <AuthContext.Provider value={value}>
    {!loading && children}
  </AuthContext.Provider>
);
}

```

설명: JWT 토큰 기반 인증, 토큰 자동 만료 처리, React Context를 활용한 전역 인증 상태 관리가 구현되어 있습니다.

4. 스케줄 CRUD API

```

const express = require('express');
const router = express.Router();
const Schedule = require('../models/Schedule');
const auth = require('../middleware/auth');
const openaiService = require('../services/openaiService');

// 인증 미들웨어를 모든 라우트에 적용
router.use(auth);

// ✅ GET: 전체 일정 조회 (자신의 일정만)
router.get('/', async (req, res) => {
  try {
    // req.user에서 userId 추출 (auth 미들웨어에서 설정됨)
    const userId = req.user.userId;

    // 사용자 자신의 일정만 조회
    const schedules = await Schedule.find({ userId });

```

```

    console.log(` ▶ 일정 개수: ${schedules.length}, 사용자: ${userId}`);
    res.json(schedules);
  } catch (err) {
    console.error('❌ GET /api/schedules 에러:', err);
    res.status(500).json({ error: err.message });
  }
});

```

// ✅ POST: 일정 등록

```

router.post('/', async (req, res) => {
  try {
    // req.user에서 userId 추출 (auth 미들웨어에서 설정됨)
    const userId = req.user.userId;

    // 요청 본문의 userId를 auth 미들웨어에서 확인된 userId로 설정
    const scheduleData = {
      ...req.body,
      userId
    };

    const schedule = new Schedule(scheduleData);
    await schedule.save();

    res.status(201).json({ message: '일정 등록 완료', schedule });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

```

// ✅ PUT: 일정 수정

```

router.put('/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const updates = req.body;
    const userId = req.user.userId;

    // 먼저 일정이 현재 로그인한 사용자의 것인지 확인
    const schedule = await Schedule.findOne({ _id: id, userId });
  }
});

```

```

    if (!schedule) {
      return res.status(404).json({ error: '일정을 찾을 수 없거나 접근 권한이 없습니다.' });
    }

    // 권한 확인 후 업데이트 진행
    const updatedSchedule = await Schedule.findByIdAndUpdate(id, updates, { new: true });

    res.json({ message: '일정 수정 완료', schedule: updatedSchedule });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// ✅ DELETE: 일정 삭제
router.delete('/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const userId = req.user.userId;

    // 먼저 일정이 현재 로그인한 사용자의 것인지 확인
    const schedule = await Schedule.findOne({ _id: id, userId });

    if (!schedule) {
      return res.status(404).json({ error: '일정을 찾을 수 없거나 접근 권한이 없습니다.' });
    }

    // 일정 삭제
    await Schedule.findByIdAndDelete(id);


    res.json({ message: '일정 삭제 완료' });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

```

설명: 사용자별 일정 CRUD (생성, 조회, 수정, 삭제) API, 권한 기반 접근 제어가 구현되어 있습니다.

5. 음성 입력 및 AI 연동

음성 데이터 수신 및 전달 API

```
//  POST: 음성 인식 텍스트 파싱만 (저장하지 않음)
router.post('/voice-parse', async (req, res) => {
  try {
    const { text } = req.body;
    if (!text) {
      return res.status(400).json({ error: '음성 인식 텍스트가 필요합니다.' });
    }

    const result = await openaiService.classifyAndExtractSchedule(text);
    if (!result.title || !result.category) {
      return res.status(400).json({ error: '시간, 일정 제목, 카테고리를 모두 말씀해 주세요.' });
    }

    // KST 시간을 그대로 사용 (변환하지 않음)
    const startTime = result.startTime;
    const endTime = result.endTime;

    console.log(`🕒 KST 시간 사용: "${startTime}"`);

    // 파싱된 결과만 반환 (저장하지 않음)
    const scheduleData = {
      title: result.title,
      categoryCode: result.category,
      startTime: startTime,
      endTime: endTime,
      type: 'general',
      priority: '보통',
      color: '#BAE1FF',
      isAllDay: result.isAllDay || false,
      description: result.description || ''
    };
  }
});
```

```

};

res.json({
  message: '음성 인식 파싱 완료',
  schedule: scheduleData
});
} catch (err) {
  console.error('음성 인식 파싱 중 오류:', err);
  res.status(500).json({ error: err.message });
}
});

```

OpenAI 연동 서비스

```

const classifyAndExtractSchedule = async (text) => {
  try {
    console.log('OpenAI API 호출 시작:', text);

    const today = getDateString(0);
    const tomorrow = getDateString(1);
    const dayAfter = getDateString(2);

    const systemPrompt = `당신은 일정을 분석하는 AI 어시스턴트입니다.
    다음 카테고리 중 하나를 선택하여 일정을 분류해주세요:
    - school (학업): 학교, 학원, 공부, 시험, 과제 관련
    - housework (가사): 청소, 빨래, 요리, 정리 등 가사 관련
    - work (업무): 회의, 프로젝트, 업무 관련
    - selfdev (자기계발): 독서, 운동, 취미, 강의 등 개인 성장 관련
    - family (가족): 가족 모임, 가족 행사, 가족 관련
    - health (건강): 병원, 건강검진, 운동, 식단 관련
    - event (행사): 모임, 파티, 축하, 기념일 등 행사 관련
    - goal (목표): 목표 달성, 계획, 리뷰 관련

    현재 날짜: ${today}

    응답은 다음 JSON 형식으로 해주세요:
    {
      "title": "일정 제목",

```

```

"startTime": "YYYY-MM-DD HH:mm",
"endTime": "YYYY-MM-DD HH:mm",
"category": "카테고리 코드",
"isAllDay": true/false
}

```

날짜 처리 규칙:

1. 날짜가 언급되지 않은 경우 `${today}` 사용
2. "오늘"은 `${today}` 사용
3. "내일"은 `${tomorrow}` 사용
4. "모레"는 `${dayAfter}` 사용

시간 처리 규칙:

1. "오전/아침"은 00:00-11:59
2. "오후"는 12:00-23:59
3. "저녁"은 18:00-23:59
4. "새벽"은 00:00-05:59
5. 시간이 언급되지 않은 경우 하루종일로 설정

예시:

```

- "오늘 오후 2시 회의" -> startTime: "${today} 14:00", endTime: "${today} 15:00"
- "내일 아침 9시 미팅" -> startTime: "${tomorrow} 09:00", endTime: "${tomorrow} 10:00"
- "모레 저녁 7시 저녁 약속" -> startTime: "${dayAfter} 19:00", endTime: "${dayAfter} 20:00"
`
;

```

```

const response = await openai.chat.completions.create({
  model: 'gpt-3.5-turbo',
  messages: [
    { role: 'system', content: systemPrompt },
    { role: 'user', content: text },
  ],
  temperature: 0.7,
  max_tokens: 500,
});

```

```

console.log('OpenAI API 응답:', response.choices[0].message.content);

let result;
try {
  result = JSON.parse(response.choices[0].message.content);
} catch (e) {
  // JSON이 아니면 에러 메시지 반환
  throw new Error('일정 정보를 추출하는데 실패했습니다. 다시 시도해주세요.');
```



```

}

if (!result.title || !result.category) {
  throw new Error('일정 제목과 카테고리를 모두 말씀해 주세요.');
```



```

}
if (!Object.keys(CATEGORY_MAPPING).includes(result.category)) {
  throw new Error('유효하지 않은 카테고리입니다.');
```



```

}

if (!result.startTime) {
  result.startTime = `${today} 00:00`;
  result.endTime = `${today} 23:59`;
  result.isAllDay = true;
}

return result;
} catch (error) {
  console.error('OpenAI API 호출 중 오류:', error);
  if (error.message.includes('JSON')) {
    throw new Error('일정 정보를 추출하는데 실패했습니다. 다시 시도해주세요.');
```



```

  }
  if (error.code === 'invalid_api_key') {
    throw new Error('OpenAI API 키가 유효하지 않습니다. 관리자에게 문의해주세요.');
```



```

  }
  throw new Error('일정 분석 중 오류가 발생했습니다: ' + error.message);
}
};

```

설명: 음성 텍스트를 OpenAI GPT로 분석하여 일정 정보 추출, 날짜/시간 자동 인식, 카테고리 분류 기능이 구현되어 있습니다.

6. 캘린더 뷰 및 드래그앤드롭

Read file: frontend/src/App.js

6. 캘린더 뷰 및 드래그앤드롭

```
// Drag & Drop
const onEventDrop = async ({ event, start, end }) => {
  const updated = { ...event, start, end };
  setEvents(prev => prev.map(ev => ev.id === event.id ? updated : ev));
  if (event._id) {
    try {
      await updateSchedule(updated);
    } catch (err) {
      console.error(err);
    }
  }
};

const onEventResize = async ({ event, start, end }) => {
  const updated = { ...event, start, end };
  setEvents(prev => prev.map(ev => ev.id === event.id ? updated : ev));
  if (event._id) {
    try {
      await updateSchedule(updated);
    } catch (err) {
      console.error(err);
    }
  }
};

// Styles
const eventStyleGetter = event => ({
  style: {
    backgroundColor: event.color,
    borderRadius: '0.5rem',
```



```

    boxShadow: '0 1px 4px rgba(0,0,0,0.1)',
    color: '#1F2937',
    padding: '4px 8px',
    fontSize: '0.875rem',
    fontWeight: 500,
    border: 'none',
    display: 'block',
    width: '100%',
  }
});

const minTime = new Date(); minTime.setHours(0, 0, 0);
const maxTime = new Date(); maxTime.setHours(23, 59, 59);

// 시간 포맷 커스터마이징
const formats = {
  eventTimeRangeFormat: () => '' // 시간 범위를 표시하지 않음
};

```

설명: React Big Calendar를 활용한 캘린더 뷰, 드래그앤드롭으로 일정 이동/크기 조정, 카테고리별 색상 표시 기능이 구현되어 있습니다.

7. 음성 입력 UI

Read file: frontend/src/App.js

```

// Speech Recognition
const startRecognition = () => {
  if (!('SpeechRecognition' in window || 'webkitSpeechRecognition' in window)) {
    console.warn('이 브라우저는 Web Speech API를 지원하지 않습니다.');
```

```

    return;
  }
  const SR = window.SpeechRecognition || window.webkitSpeechRecognition;
  const rec = new SR();
  rec.lang = 'ko-KR';
  rec.continuous = true;

```

```

rec.interimResults = true;
rec.maxAlternatives = 1;

let finalTranscript = '';
let silenceTimer = null;

rec.onstart = () => {
  console.log('음성 인식 시작됨');
  setRecording(true);
  setTranscript('');
  finalTranscript = '';
};

rec.onresult = event => {
  let interimTranscript = '';

  for (let i = event.resultIndex; i < event.results.length; i++) {
    const transcript = event.results[i][0].transcript;
    if (event.results[i].isFinal) {
      finalTranscript += transcript;
    } else {
      interimTranscript += transcript;
    }
  }
}

console.log('중간 결과:', interimTranscript);
console.log('누적 결과:', finalTranscript);

setTranscript(finalTranscript || interimTranscript);

if (silenceTimer) clearTimeout(silenceTimer);
silenceTimer = setTimeout(() => {
  if (finalTranscript) {
    console.log('무음 감지, 음성 인식 종료');
    rec.stop();
  }
}, 1000);
};

```

```

rec.onerror = err => {
  console.error('음성 인식 에러:', err);
  setError('음성 인식 중 오류가 발생했습니다.');
```

setShowModal(true);

```

  setRecording(false);
  if (silenceTimer) clearTimeout(silenceTimer);
};

rec.onend = () => {
  console.log('음성 인식 종료됨');
  console.log('최종 인식 텍스트:', finalTranscript);
  setRecording(false);
  if (silenceTimer) clearTimeout(silenceTimer);

  if (finalTranscript) {
    handleVoiceInputSchedule(finalTranscript);
  }
};

recognitionRef.current = rec;
rec.start();
};
const stopRecognition = () => recognitionRef.current?.stop();

```

설명: Web Speech API를 활용한 한국어 음성 인식, 실시간 음성 텍스트 변환, 무음 감지 자동 종료 기능이 구현되어 있습니다.

8. 스케줄 요약 및 TTS 브리핑

```

// 일정 요약(브리핑) - 오늘/내일/이번주/이번달
router.get('/briefing', async (req, res) => {
  try {
    const { type = 'today' } = req.query;
    const today = new Date();
    let startDate, endDate, periodText;

    console.log(`July17 브리핑 요청: type=${type}, 현재시간=${today}`);
  }
}

```

```

switch (type) {
  case 'tomorrow':
    const tomorrow = new Date(today);
    tomorrow.setDate(today.getDate() + 1);
    startDate = new Date(tomorrow.getFullYear(), tomorrow.getMonth(), tomorrow.getDate(), 0, 0, 0);
    endDate = new Date(tomorrow.getFullYear(), tomorrow.getMonth(), tomorrow.getDate(), 23, 59, 59);
    periodText = '내일';
    break;

  case 'week':
    const startOfWeek = new Date(today);
    startOfWeek.setDate(today.getDate() - today.getDay()); // 일요일부터
    const endOfWeek = new Date(startOfWeek);
    endOfWeek.setDate(startOfWeek.getDate() + 6); // 토요일까지
    startDate = new Date(startOfWeek.getFullYear(), startOfWeek.getMonth(), startOfWeek.getDate(), 0, 0, 0);
    endDate = new Date(endOfWeek.getFullYear(), endOfWeek.getMonth(), endOfWeek.getDate(), 23, 59, 59);
    periodText = '이번 주';
    break;

  case 'month':
    startDate = new Date(today.getFullYear(), today.getMonth(), 1, 0, 0, 0);
    endDate = new Date(today.getFullYear(), today.getMonth() + 1, 0, 23, 59, 59);
    periodText = '이번 달';
    break;

  default: // 'today'
    startDate = new Date(today.getFullYear(), today.getMonth(), today.getDate(), 0, 0, 0);
    endDate = new Date(today.getFullYear(), today.getMonth(), today.getDate(), 23, 59, 59);
    periodText = '오늘';
}

```

```

const userId = req.user.userId;

const schedules = await Schedule.find({
  userId,
  startTime: { $gte: startDate, $lte: endDate }
}).sort({ startTime: 1 });

if (schedules.length === 0) {
  return res.json({ message: `${periodText}은 등록된 일정이 없습니다.` });
}

let message = `${periodText} 일정은 총 ${schedules.length}건입니다.\n`;

schedules.forEach((s, i) => {
  const date = type === 'today' || type === 'tomorrow' ? '' : `${s.startTime.getMonth() + 1}월 ${s.startTime.getDate()}일`;
  const time = s.isAllDay ? '하루종일' : s.startTime.toTimeString().slice(0, 5);
  const lineMessage = `${i + 1}. ${date}${time} - ${s.title}`;
  message += lineMessage + '\n';
});

res.json({ message });
} catch (error) {
  console.error('일정 브리핑 중 오류:', error);
  res.status(500).json({ error: '일정 브리핑에 실패했습니다.' });
}
});

```

TTS 브리핑 기능 (프론트엔드)

```

// TTS로 읽어주기
if ('speechSynthesis' in window) {
  const utter = new window.SpeechSynthesisUtterance(message);
  utter.lang = 'ko-KR';
  window.speechSynthesis.speak(utter);
}

```

```
setTranscript('');  
setIsLoading(false);  
return;
```

설명: 음성 명령으로 일정 브리핑 요청, 기간별(오늘/내일/주간/월간) 일정 요약, TTS를 통한 음성 브리핑 기능이 구현되어 있습니다.

9. 데이터베이스 연동

스케줄 모델

```
const mongoose = require('mongoose');  
  
const scheduleSchema = new mongoose.Schema({  
  userId: {  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'User', // user 테이블과 연결  
    required: true,  
  },  
  title: {  
    type: String,  
    required: true,  
  },  
  description: {  
    type: String,  
    default: '',  
  },  
  // 일정 종류에 따라 시간 정보 다르게 입력됨  
  startTime: {  
    type: Date,  
    required: true,  
  },  
  endTime: {  
    type: Date,  
    required: true,  
  },  
  dueDate: Date, // 기한 일정
```

```

isRepeating: {
  type: Boolean,
  default: false,
},
repeatPattern: {
  type: String,
  enum: ['DAILY', 'WEEKLY', 'MONTHLY'],
},

categoryCode: {
  type: String,
  enum: ['school', 'housework', 'work', 'selfdev', 'family', 'health', 'event', 'goal'],
  required: true,
},

priority: {
  type: String,
  enum: ['낮음', '보통', '높음'],
  default: '보통',
},

type: {
  type: String,
  enum: ['general', 'meeting', 'task', 'reminder'],
  default: 'general',
},

// 새로 추가된 색상 필드
color: {
  type: String,
  default: '#BAE1FF',
},

isCompleted: {
  type: Boolean,
  default: false,

```

```

    },

    isAllDay: {
      type: Boolean,
      default: false,
    }
  }, {
    timestamps: true // createdAt, updatedAt 자동 추가
  });

module.exports = mongoose.model('Schedule', scheduleSchema);

```

설명: MongoDB Mongoose를 활용한 스케줄 데이터 모델, 사용자별 일정 관리, 카테고리/우선순위/색상 등 다양한 속성 지원이 구현되어 있습니다.

10. 로그아웃 기능

백엔드 API 연동

```

// 회원 탈퇴
deleteAccount: async () => {
  const token = getToken();
  console.log('토큰 확인:', token ? 'Token exists' : 'No token');

  try {
    const response = await fetch(`${API_URL}/auth/delete`, {
      method: 'DELETE',
      headers: {
        ...authHeader(),
        'Content-Type': 'application/json'
      },
    });
  };

  if (!response.ok) {
    console.error('Delete account error status:', response.status);
    console.error('Delete account error statusText:', response.statusText);
  }
}

```



```

    return handleResponse(response);
  } catch (error) {
    console.error('Delete account error:', error);
    throw error;
  }
},

```

프론트엔드 로그아웃 구현

```

const login = (userData) => {
  localStorage.setItem('token', userData.token);
  setToken(userData.token);
  setCurrentUser(jwtDecode(userData.token));
};

const logout = () => {
  localStorage.removeItem('token');
  setToken(null);
  setCurrentUser(null);
};

```

설명: localStorage에서 토큰 제거, 인증 상태 초기화, 자동 로그인 페이지 리디렉션 기능이 구현되어 있습니다.

기능 요약

이 프로젝트에는 다음과 같은 주요 기능들이 구현되어 있습니다:

구현된 기능들

1. **호스팅 & 서버:** HTTPS 서버, SSL 인증서, MongoDB 연동
2. **인증 시스템:** JWT 기반 로그인/로그아웃, 사용자 권한 관리
3. **캘린더 뷰:** React Big Calendar, 드래그앤드롭, 색상 표시
4. **음성 인식:** Web Speech API, 한국어 음성 텍스트 변환
5. **AI 연동:** OpenAI GPT로 일정 자동 분석 및 생성
6. **스케줄 관리:** CRUD API, 카테고리별 분류, 우선순위 설정

7. 브리핑 기능: 음성 명령으로 일정 요약, TTS 음성 출력

8. 데이터베이스: MongoDB로 사용자별 데이터 관리

주요 특징

- 음성 기반 인터페이스: "오늘 오후 2시 회의"라고 말하면 자동으로 일정 생성
- AI 스케줄 분석: OpenAI로 자연어를 구조화된 일정 데이터로 변환
- 실시간 브리핑: "오늘 일정 알려줘"라고 하면 TTS로 음성 안내
- 직관적 UI: 드래그앤드롭으로 일정 이동, 카테고리별 색상 구분
- 사용자별 격리: JWT 토큰으로 개인 일정 보안 관리

이 프로젝트는 음성 인식과 AI를 활용한 차세대 스케줄 관리 시스템으로, 사용자가 자연스러운 음성 명령으로 일정을 관리할 수 있는 혁신적인 기능들을 제공합니다.