# ConvEnviTest 프로젝트 요약

## 🏷 프로젝트 개요

ConvEnviTest는 환경 테스트 장비를 제어하고 모니터링하는 웹 기반 시스템입니다. 이 시스템은 산업용 장비의 자동화된 테스트를 위한 완전한 솔루션을 제공합니다.

## 🏗 기술 스택

백엔드

- **Node.js**: 서버 런타임
- **WebSocket**: 실시간 통신
- **SerialPort**: 시리얼 통신 (장비 제어)
- **ws**: WebSocket 라이브러리

프론트엔드

- **Next.js**: React 프레임워크
- **React**: UI 라이브러리
- **Material-UI**: UI 컴포넌트
- **Tailwind CSS**: 스타일링
- **TypeScript**: 타입 안전성

## 📁 프로젝트 구조

```
convEnviTest/
├── 📄 README.md                    # 메인 사용설명서
├── 📄 INSTALL_GUIDE.md             # 상세 설치 가이드
├── 📄 PROJECT_SUMMARY.md           # 이 파일
├── 📄 POWER_SWITCH_IMPLEMENTATION.md # 전원 스위치 구현 문서
├── 🐳 docker-compose.yml           # Docker Compose 설정
├── 📄 install.sh                   # Linux/Mac 설치 스크립트
├── 📄 install.bat                  # Windows 설치 스크립트
│
├── 🖥 server/                      # 백엔드 서버
│   ├── 📄 backend-websocket-server.js # 메인 서버
│   ├── 📄 package.json             # 서버 의존성
│   ├── 📄 Dockerfile               # 백엔드 Docker 설정
│   ├── 📄 ReadChamber.js           # 챔버 온도 읽기
│   ├── 📄 ReadVolt.js              # 전압 측정
│   ├── 📄 SetVolt.js               # 전압 설정
│   ├── 📄 SelectDevice.js          # 릴레이 제어
│   ├── 📄 RunTestProcess.js        # 자동 테스트 프로세스
│   └── 📄 *.json                   # 설정 저장 파일들
│
└── 🌐 nextjs/                      # 프론트엔드 애플리케이션
    ├── 📄 package.json             # 프론트엔드 의존성
```

```
├── 📄 Dockerfile                # 프론트엔드 Docker 설정
├── 📄 next.config.mjs           # Next.js 설정
├── 📄 tailwind.config.js        # Tailwind CSS 설정
├── 📄 tsconfig.json             # TypeScript 설정
│
├── 📁 pages/                    # Next.js 페이지
│   ├── 📄 index.js              # 메인 페이지
│   ├── 📄 _app.js               # 앱 설정
│   └── 📄 _document.js          # 문서 설정
│
├── 📁 components/               # React 컴포넌트들
│   ├── 📁 PowerButton/          # 전원 스위치
│   ├── 📁 RelayOn/              # 릴레이 제어
│   ├── 📁 SetVolt/              # 전압 설정
│   ├── 📁 ReadVolt/             # 전압 측정
│   ├── 📁 ReadChamber/          # 온도 모니터링
│   ├── 📁 TestProcess/          # 테스트 프로세스
│   ├── 📁 SystemSet/            # 시스템 설정
│   ├── 📁 UsbPortSelect/        # USB 포트 선택
│   ├── 📁 DeviceSelect/         # 장비 선택
│   ├── 📁 TestResult/           # 테스트 결과
│   ├── 📁 LogoImage/            # 로고 이미지
│   ├── 📁 GroupImage/           # 그룹 이미지
│   ├── 📁 TextButton/           # 텍스트 버튼
│   ├── 📁 TestSystem/           # 테스트 시스템
│   ├── 📁 power-table/          # 전력 테이블
│   ├── 📁 high-temp-setting-panel/ # 고온 설정
│   ├── 📁 low-temp-setting-panel/  # 저온 설정
│   ├── 📁 delay-settings-panel/    # 딜레이 설정
│   ├── 📁 OutVoltSettingPanel/     # 출력 전압 설정
│   ├── 📁 ChannelVoltageSettings/  # 채널 전압 설정
│   └── 📁 OptionSet1/           # 옵션 설정
│
├── 📁 styles/                   # CSS 스타일
│   ├── 📄 globals.css           # 전역 스타일
│   └── 📄 Home.module.css       # 홈 페이지 스타일
│
├── 📁 lib/                      # 유틸리티 라이브러리
│   ├── 📄 parsePowerData.ts     # 전력 데이터 파싱
│   └── 📄 useIsClient.ts        # 클라이언트 훅
│
└── 📁 data/                     # 데이터 파일들
    ├── 📄 data1.txt             # 테스트 데이터
    └── 📄 data1_1.txt           # 추가 테스트 데이터
```

## 🔧 주요 기능

### 1. 장비 제어

- **릴레이 제어**: 10개 채널의 릴레이 ON/OFF 제어
- **전압 설정**: 각 채널별 전압값 설정 (0-30V)
- **온도 제어**: 챔버 온도 설정 및 모니터링

- **시리얼 통신**: Modbus RTU 프로토콜을 통한 장비 통신

### 2. 자동화된 테스트

- **자동 테스트 프로세스**: 설정된 조건에 따른 자동 테스트 실행
- **실시간 모니터링**: 전압, 온도, 테스트 결과 실시간 표시
- **결과 저장**: 테스트 결과를 JSON 파일로 저장

### 3. 웹 인터페이스

- **반응형 디자인**: 모바일 및 데스크톱 지원
- **실시간 업데이트**: WebSocket을 통한 실시간 데이터 통신
- **사용자 친화적 UI**: Material-UI 기반의 직관적인 인터페이스

### 4. 설정 관리

- **설정 저장**: 모든 설정값을 JSON 파일로 저장
- **설정 복원**: 서버 재시작 시 설정 자동 복원
- **실시간 설정 변경**: 서버 재시작 없이 설정 변경 가능

## 🔌 하드웨어 지원

### 시리얼 통신

- **포트**: COM1-COM10 (Windows), /dev/ttyUSB* (Linux/Mac)
- **프로토콜**: Modbus RTU
- **통신 속도**: 9600 bps
- **데이터 비트**: 8
- **패리티**: None
- **정지 비트**: 1

### 릴레이 명령어

시스템은 다음 Modbus 명령어를 사용합니다:

| 릴레이 | ON 명령어 | OFF 명령어 |
|---|---|---|
| #1 | 010600010100D99A | 010600010200D96A |
| #2 | 010600020100299A | 010600020200296A |
| #3 | 010600030100785A | 01060003020078AA |
| #4 | 010600040100C99B | 010600040200C96B |
| #5 | 010600050100985B | 01060005020098AB |
| #6 | 020600010100D9A9 | 020600010200D959 |
| #7 | 02060002010029A9 | 0206000202002959 |
| #8 | 0206000301007869 | 0206000302007899 |

| 릴레이 | ON 명령어 | OFF 명령어 |
|---|---|---|
| #9 | 020600040100C9A8 | 020600040200C958 |
| #10 | 0206000501009868 | 0206000502009898 |

## 📊 데이터 관리

### 설정 파일

모든 설정은 JSON 파일로 관리됩니다:

- `delay_settings.json`: 딜레이 설정
- `device_states.json`: 장비 상태
- `high_temp_settings.json`: 고온 설정
- `low_temp_settings.json`: 저온 설정
- `product_input.json`: 제품 정보
- `usb_port_settings.json`: USB 포트 설정
- `out_volt_settings.json`: 출력 전압 설정
- `channel_voltages.json`: 채널 전압 설정

### 데이터 형식

```
{
  "delay_settings": {
    "onDelay": 1000,
    "offDelay": 2000,
    "cycleNumber": 5
  },
  "device_states": {
    "relay1": false,
    "relay2": true,
    "relay3": false
  }
}
```

## 🚀 배포 옵션

### 1. 로컬 개발 환경

- Node.js 직접 설치
- 수동 의존성 관리
- 개발 서버 실행

### 2. 자동 설치 스크립트

- Windows: `install.bat`
- Linux/Mac: `install.sh`
- 자동 의존성 설치 및 실행 스크립트 생성

### 3. Docker 컨테이너

- `docker-compose.yml` 사용
- 격리된 환경에서 실행
- 쉬운 배포 및 관리

## 🔒 보안 고려사항

### 네트워크 보안

- WebSocket 연결은 로컬 네트워크에서만 실행
- 외부 접근을 위한 방화벽 설정 필요

### 하드웨어 보안

- 시리얼 포트 접근 권한 관리
- 장비 제어 명령어 검증
- 안전한 종료 프로세스

## 📈 성능 최적화

### 프론트엔드 최적화

- Next.js SSR/SSG 활용
- 이미지 최적화 (WebP, lazy loading)
- 코드 스플리팅 및 동적 임포트

### 백엔드 최적화

- WebSocket 연결 풀 관리
- 시리얼 통신 최적화
- 메모리 사용량 모니터링

## 🧪 테스트 전략

### 단위 테스트

- React 컴포넌트 테스트
- Node.js 모듈 테스트
- WebSocket 통신 테스트

### 통합 테스트

- 전체 시스템 테스트
- 하드웨어 연동 테스트
- 성능 테스트

## 📝 개발 가이드라인

### 코드 스타일

- TypeScript 사용
- ESLint 규칙 준수
- Prettier 포맷팅

### 커밋 메시지

- Conventional Commits 사용
- 명확한 변경 사항 설명
- 이슈 번호 연결

### 문서화

- JSDoc 주석 사용
- README 파일 유지보수
- API 문서 작성

## 🤝 기여 가이드

### 개발 환경 설정

1. 프로젝트 클론
2. 의존성 설치
3. 개발 서버 실행
4. 테스트 실행

### 코드 리뷰

- Pull Request 필수
- 코드 리뷰 승인 후 머지
- 테스트 커버리지 확인

## 📞 지원 및 문의

### 문제 해결

1. 로그 확인
2. 버전 호환성 검사
3. 하드웨어 연결 상태 확인
4. 네트워크 설정 확인

### 문서 참조

- `README.md`: 기본 사용법
- `INSTALL_GUIDE.md`: 상세 설치 가이드
- `POWER_SWITCH_IMPLEMENTATION.md`: 전원 스위치 구현

---

**마지막 업데이트**: 2025년 08월

이 문서는 ConvEnviTest 프로젝트의 전체적인 구조와 기능을 설명합니다. 자세한 사용법은 각 섹션의 링크된 문서를 참조하세요.