「S-개발자 4기」 개발자 매뉴얼

DIRECT P2P

P2P 통신 영상전송 / 편집 서비스

개발자 매뉴얼

팀명	3조 E둘I들
팀원	김민수, 이수진, 최하경

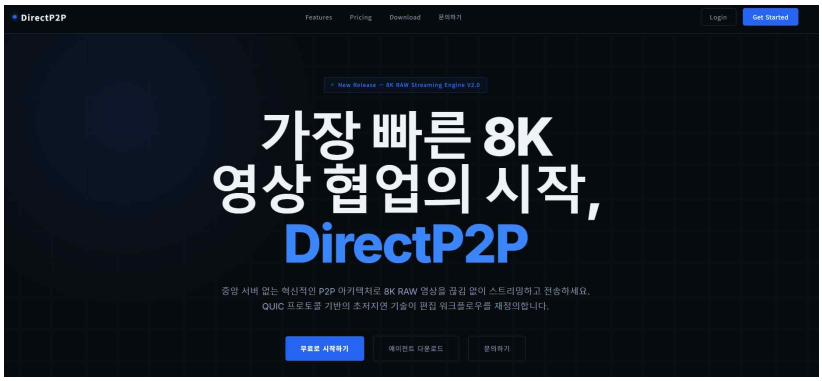
2026년 4월 3일

〈 목 차 〉

Ⅰ. 시스템 개요

1.1. 서비스 소개 및 목적

DirectP2P 는 UDP 홀펀칭 (UDP Hole Punching) 기술을 기반으로, NAT(Network Address Translation)환경에 놓인 두 클라이언트가 중앙 서버를 경유하지 않고 직접 P2P 연결을 수립할 수 있도록 설계된 고속 콘텐츠 전송 및 즉석 편집 서비스이다.



기존의 파일 전송 서비스는 중앙 서버를 통해 데이터를 중계하는 방식을 채택하고 있어,전송 속도가 서버 대역폭에 종속되고 대용량 파일을 전송할 때 병목 현상이 발생한다. DirectP2P는 이 문제를 해결하기 위해 STUN/TURN 프로토콜과 QUIC 기반 전송 계층을 결합하여, 피어 간 직접 연결이 성립될 경우 이론 회선 속도의 80% 이상의 전송 성능을 제공한다. 홀펀칭이 실패하는 엄격한 NAT 환경(Symmetric NAT 등)에서는 TURN 서버를 통한 릴레이 폴백(Relay Fallback)으로 자동 전환되어, 어떤 네트워크 환경에서든 연결 가능성을 보장한다. 또한, 전송 중인 영상에 대해 컷편집 및 프리뷰가 가능한 On-the-fly 편집 기능을 제공하여, 영상 협업 워크플로우를 단일 서비스 내에서 처리할 수 있게 한다.

1.2. 주요 기능 요약

○ **P2P 직접 연결 :** STUN 서버를 통해 각 클라이언트의 공인 IP/Port 를 파악하고 UDP 홀펀칭으로 NAT를 투과하여 두 피어 간의 직접 연결을 수립한다. 홀펀칭 실패 시엔 TURN 릴레이로 자동 전환되며, 이후 P2P 연결이 복구될 경우 다시 직접 연결로 업그레이드 된다.

○ **QUIC 기반 고속 전송 :** P2P 연결 위에 QUIC 프로토콜을 적용하여 멀티플렉싱 스트림, 혼잡 제어, 손실 복구, TLS 암호화를 제공한다. Wi-Fi - LTE 간의 전환 등 네트워크 환경이 바뀌어도 Connection Migration으로 연결이 유지된다.

○ **종단 간 보안 전송 :** 데이터는 AES-256-GCM으로 암호화되며, SHA-256 체크섬으로 무결성을 검증한다. TURN 자격증명은 HMAC-SHA256 기반의 시간 제한 임시 토큰으로 발급된다.

○ **On-the-fly :** 영상 편집 전송이 완료되기 전에도 수신중인 영상을 미리 확인, 타임라인 기반의 컷편집을 가능하게 한다.

1.3. 대상

해당 문서는 DirectP2P 시스템을 직접 구축, 운영, 확장하는 개발자를 대상으로 한다.

○ **백엔드 개발자 :** Go 언어로 작성된 API Server 및 Local Agent 의 내부 구조와 API 명세를 파악하고 기능을 추가하거나 유지보수하려는 개발자
○ **프론트엔드 개발자 :** React 기반 Web Frontend의 화면 구성, 로컬 에이전트 및 API Server와의 통신 방식을 이해하고 UI를 개발하려는 개발자
○ **인프라DevOps 엔지니어:** Nginx, coturn, Jenkins, GitHub Actions로 구성된 배포 파이프라인을 설정하고 운영하려는 엔지니어
○ **시스템 통합 담당자:** DirectP2P의 구성 요소(Static Server, API Server, TURN Server, Local Agent, Web Frontend)를 연동하여 전체 서비스를 구성하려는 담당자

1.4. 문서 구성 안내

이 매뉴얼은 DirectP2P 시스템의 개발 환경 구성부터 각 컴포넌트의 상세 동작 원리, API 명세, 배포 절차까지를 순서대로 다룬다. 처음 합류한 개발자라면 1장부터 순서대로 읽을 것을 권장하며, 특정 컴포넌트의 명세만 필요한 경우 해당 챕터로 바로 이동하여 참조할 수 있다.

챕터	내용
1장. 시스템 개요	서비스 목적, 주요 기능, 대상 독자, 문서 구성
2장. 시작하기 전에	개발 환경 요구사항, 필요 계정 및 권한, 사전 지식
3장. 아키텍처	전체 시스템 구성도, 컴포넌트 역할, 연결 흐름
4장. 컴포넌트별 상세	Static Server, API Server, TURN Server, Local Agent, Web Frontend
5장. API 명세	REST API 엔드포인트, WebSocket 이벤트, 로컬 에이전트 API
6장. 배포 및 CI/CD	GitHub Actions, Jenkins 파이프라인, Docker 배포
7장. 오류 코드 및 트러블 슈팅	오류 코드 정의, 장애 대응 가이드
부록	SBOM, 라이선스, 외부 참조

II. 시작하기 전에

2.1 요구사항

해당 서비스의 개발 및 유지 보수 시에 필요한 운영체제, 언어 및 런타임, 주요 의존 소프트웨어와 네트워크 환경은 다음과 같다.

○ 운영 체제

구분	지원환경
Local Agent (개발)	Windows 10+, macOS 11+, Ubuntu 20.04+ / Debian 10+
서버 컴포넌트	Linux 계열 (Ubuntu 20.04+ 권장), Docker 지원 환경
웹 브라우저	Chrome 90+, Firefox 88+, Edge 90+, Safari 14+

○ 언어 및 런타임

항목	버전	용도
Go	1.22.0 이상	API Server, Local Agent 빌드
Node.js	18 LTS 이상	Web Frontend 빌드
npm	9.x 이상	프론트엔드 패키지 관리

○ 주요 의존 소프트웨어

소프트웨어	버전	용도
NginX	1.25.3	정적 파일 서빙 및 리버스 프록시
coturn	4.6.2	TURN 서버
ffmpeg	6.1.1	영상 트랜스코딩
SQLite	3.45.1	로컬 및 서버 데이터베이스
OpenSSL	3.0.13	암호화 라이브러리
Docker	24.x 이상	컨테이너 기반 배포
Jenkins	2.452.1	CI/CD 파이프라인

○ 네트워크 환경

서버 컴포넌트 운영 시엔 아래 포트가 개방되어 있어야 한다.

포트	프로토콜	용도
80, 443	TCP	Nginx HTTP/HTTPS
3478	UDP/	STUN/TURN
5349	TCP	TURN over TLS
17432	TCP (localhost)	Local Agent 로컬 HTTP API

2.2 필요한 계정 및 권한

GitHub DirectP2P는 메인 리포지토리와 하위 컴포넌트 리포지토리를 Git Submodule로 관리한다. 개발에 참여하려면 GitHub 계정과 해당 조직(E2IDLE) 리포지토리에 대한 접근 권한이 필요하다. CI/CD 자동화를 위한 GitHub Actions 워크플로우가 각 리포지토리에 설정되어 있으므로, Actions 실행 권한도 확인해야 한다.

○ **메인 리포지토리 :** https://github.com/E2IDLE/DirectP2P
○ **필요 권한 :** 리포지토리 Read(필수), Write(기여자), Actions 실행 권한

서버 접근 API Server, TURN Server, Static Server가 배포되는 운영 서버에 대한 SSH 접근 권한이 필요하다. 배포 스크립트(install.sh, install.ps1)는 SCP/SFTP를 통해 빌드 산출물을 서버로 전송하므로, 해당 서버에 대한 쓰기 권한을 사전에 확인해야 한다.

Jenkins 빌드 및 배포 파이프라인은 Jenkins를 통해 실행됩니다. Jenkins 관리자 계정 또는 파이프라인 실행 권한이 있는 계정이 필요하다. Webhook 연동을 위해 Jenkins 서버가 GitHub으로부터 인바운드 요청을 수신할 수 있는 네트워크 환경이어야 한다.
coturn 설정 TURN 서버(coturn)의 시간 제한 자격증명 검증에는 API Server와 공유하는 secret_key가 필요하다. 이 값은 .env 또는 config.yaml에 환경 변수로 관리되며, 운영 환경에서는 반드시 안전하게 보관하고 소스 코드에 직접 포함하지 않아야 한다.

2.3 사전 지식 수준

Jenkins 빌드 및 배포 파이프라인은 Jenkins를 통해 실행됩니다. Jenkins 관리자 계정 또는 파이프라인 실행 권한이 있는 계정이 필요하다. Webhook 연동을 위해 Jenkins 서버가 GitHub으로부터 인바운드 요청을 수신할 수 있는 네트워크 환경이어야 한다.

- **공통 (모든 기여자)**
  - Git 기본 사용법 및 Git Submodule 개념
  - HTTP/HTTPS, REST API, WebSocket의 기본 동작 원리
  - Docker 컨테이너의 기본 개념 (이미지 빌드, 컨테이너 실행)

- **백엔드 개발자 (Go)**
  - Go 언어 문법 및 표준 라이브러리 활용
    (goroutine, channel, context 등)
  - Gin 프레임워크를 이용한 REST API 개발 경험
  - UDP 소켓 프로그래밍 기초
  - NAT, STUN, TURN, ICE, UDP 홀펀칭의 동작 원리
  (이 프로젝트의 핵심 기술이므로 RFC 5389(STUN), RFC 5766(TURN), RFC 5245(ICE)를 사전에 숙지할 것을 '강하게 권장)
  - QUIC 프로토콜의 기본 개념 (RFC 9000)
  - TLS 1.3 및 AES-256-GCM 암호화 기초

- **프론트엔드 개발자 (React)**
  - React 18 기반 SPA 개발 경험 (Hooks, 컴포넌트 설계)
  - WebSocket 클라이언트 연동 경험
  - REST API 비동기 호출 (fetch / axios)
  - 로컬 HTTP API(localhost:17432)와 통신하는 구조 이해

- **인프라/DevOps 엔지니어**
  - Nginx 설정 (리버스 프록시, SPA 라우팅, HTTPS 적용)
  - GitHub Actions 워크플로우 작성 경험
  - Jenkins 파이프라인 구성 및 Webhook 연동
  - Linux 서버 환경에서의 서비스 관리 (systemd, 방화벽 설정)
  - coturn 설정 및 운영 경험이 있으면 유리함

2.3 WBS

모든 작업은 WBS에 기록하고 담당자를 지정, 시작일시와 마감일시를 정한 후 작업 달성량을 WBS 시트에 기록하는 방식으로 진행한다. 구분은 크게 기획 및 설계, 개발, 산출물 정리의 단계로 구성되어 있으며 분야는 인프라/공통과 프론트엔드, 백엔드, 에이전트로 구분한다.
작업에 대한 상세 내용 혹은 다른 말은 비고란에 작성한다.

DirectP2P — WBS (Work Breakdown Structure)

3조 E등1둘 · 개발 인원 3명 · 마감: 아번주 금요일 (2026-04-03)

구분	작업 제목		상세 작업	비고	담당자	시작 일시	마감 일시	작업 달성량 (%)
기획 및 설계	개발환경 세팅 및 기획/설계	인프라/공통	환경변수 & 설정 파일 관리	config.yaml, .env 분리		2026-03-26 9:00	2026-03-26 18:00	100
		인프라/공통	모노레포 & 서브모듈 구조	git submodule 설정		2026-03-26 9:00	2026-03-26 18:00	100
		인프라/공통	GitHub Actions CI 워크플로우	submodule_updated 이벤트		2026-03-26 9:00	2026-03-27 18:00	100
		인프라/공통	Nginx 설정	SPA fallback, HTTPS		2026-03-27 9:00	2026-03-28 18:00	0
		인프라/공통	coturn 설정	HMAC-SHA256 인증, 포트 설정		2026-03-28 9:00	2026-03-29 18:00	0
		인프라/공통	API 명세서 작성	openapi.yaml 작성 및 명세서 문서화		2026-03-26 9:00	2026. 3. 26	100
		인프라/공통	API 명세서 최신화			2026-03-31 9:00	2026-04-02 18:00	100
		인프라/공통	기획서 작성	기획서 작성		2026-03-26 9:00	2026-03-26 18:00	100
		인프라/공통	WBS 작성	WBS 작성		2026. 4. 2	2026. 4. 2	100
		프론트엔드	기술 스택 선정	react.js		2026-03-26 9:00	2026-03-26 18:00	100
		백엔드	기술 스택 선정	go		2026-03-26 9:00	2026-03-26 18:00	100
		에이전트	기술 스택 선정	go		2026-03-26 9:00	2026-03-26 18:00	100
		인프라/공통	Git 커밋 컨벤션 정의			2026-03-26 9:00	2026-03-27 18:00	100
		인프라/공통	Git 브랜치 전략 정의			2026-03-26 9:00	2026-03-27 18:00	100
		인프라/공통	프로젝트 디렉터리 구조 설정			2026-03-26 9:00	2026-03-27 18:00	100
		인프라/공통	프로젝트 생성			2026-03-26 9:00	2026-03-27 18:00	100
		인프라/공통	에러 코드 정의 문서화	E1001~E6001 팀 공유		2026-03-31 9:00	2026-04-02 18:00	90
개발	회원가입 및 로그인	프론트엔드	랜딩 페이지		이수진	2026. 3. 27 19:00	2026. 3. 27 20:00	100
		프론트엔드	요금제 페이지		이수진	2026. 3. 27 19:00	2026. 3. 27 20:00	100
		프론트엔드	회원가입/로그인 페이지	API 연동 포함	이수진	2026. 3. 27 20:00	2026. 3. 27 22:00	100
		백엔드	회원가입 API	POST /auth/register	최하경	2026-03-26 9:00	2026-03-27 18:00	100
		백엔드	로그인 API	POST /auth/login, Token 발급	최하경	2026-03-26 9:00	2026-03-27 18:00	100
		백엔드	로그아웃 API	POST /auth/logout	최하경	2026-03-27 9:00	2026-03-27 18:00	100
	에이전트 & Websocket	프론트엔드	에이전트 설치 가이드 페이지	플랫폼별 다운로드 링크	이수진	2026. 3. 27 20:00	2026. 3. 27 22:00	100
		프론트엔드	WebSocket 연결 & 알림		이수진	2026. 4. 1 17:00	2026. 4. 1 18:00	100
		프론트엔드	에이전트 감지 & 연동		이수진	2026-03-29 9:00	2026-03-30 18:00	0
		백엔드	에이전트 등록/목록 API	POST/GET /users/me/agents	최하경	2026-03-27 9:00	2026-03-28 18:00	100
		백엔드	WebSocket 서버	GET /ws, 이벤트 Push	최하경	2026-03-29 9:00	2026-03-31 18:00	100
		에이전트	로컬 HTTP API 서버 구동		김민수			
	세션 연결 관련	백엔드	세션 생성 API	POST /sessions, 초대 코드 발급	최하경	2026-03-28 9:00	2026-03-29 18:00	100
		백엔드	세션 참가 API	POST /sessions/:id/join	최하경	2026-03-28 9:00	2026-03-29 18:00	100
		백엔드	세션 상태 조회/삭제 API	GET/DELETE /sessions/:id	최하경	2026-03-28 9:00	2026-03-29 18:00	100
		프론트엔드	연결 단계 시각화	STUN→홀펀칭 스텝 인디케이터	이수진			
		백엔드	세션 이력 조회 API	GET /sessions/history	최하경	2026-03-31 9:00	2026-04-01 18:00	100
	Candidate / Turn / 인증 관련	백엔드	Candidate 등록/조회 API	POST/GET /sessions/:id/candidates	최하경	2026-03-29 9:00	2026-03-30 18:00	100
		백엔드	TURN 자격증명 발급 API	HMAC-SHA256 임시 자격증명	최하경	2026-03-29 9:00	2026-03-30 18:00	100
	에이전트	에이전트	STUN 요청 & 공인 주소 획득	Google STUN, Mapped Address	김민수	2026-03-26 9:00	2026-03-28 18:00	100
		에이전트	Candidate 수집 & 등록	POST /sessions/:id/candidates	김민수	2026-03-27 9:00	2026-03-29 18:00	30
		에이전트	UDP 홀펀칭 구현	동시 패킷 송출, 성공 판정	김민수	2026-03-28 9:00	2026-03-31 18:00	50
		에이전트	TURN Fallback 전환	5초 타임아웃 시 릴레이 전환	김민수	2026-03-29 9:00	2026-03-31 18:00	0
		에이전트	파일 청크 분할 전송	64KB, AES-256-GCM 암호화	김민수	2026-03-29 9:00	2026-04-01 18:00	0
		에이전트	청크 수신 & 재조립	NACK 재요청, 순서 재조립	김민수	2026-03-29 9:00	2026-04-01 18:00	0
		에이전트	SHA-256 무결성 검증	전체 수신 후 체크섬 검증	김민수	2026-03-30 9:00	2026-04-01 18:00	0
		에이전트	전송 진행률 API 제공	GET /transfer/progress	김민수	2026-03-30 9:00	2026-04-02 18:00	0
		에이전트	ffmpeg 연동 (영상 스트리밍)	fMP4 트랜스코딩 → HTTP 스트림	김민수	2026-03-31 9:00	2026-04-02 18:00	0
		에이전트	ffmpeg 연동 (영상 컷편집)	타임스탬프 기반 실제 영상 슬라이싱	김민수	2026-03-31 9:00	2026-04-02 18:00	0
		에이전트	NAT Keepalive 패킷 전송	30초 주기	김민수	2026-04-01 9:00	2026-04-02 18:00	0
		에이전트	P2P 재연결 처리	최대 3회 자동 재시도	김민수	2026-04-01 9:00	2026-04-03 12:00	0
		에이전트	Windows 설치 패키지	.exe	김민수	2026-04-02 9:00	2026-04-03 12:00	100
		에이전트	Windows 설치 패키지 서비스 등록	시스템 서비스 등록	김민수	2026-04-02 9:00	2026-04-03 12:00	0
	화면 구현	프론트엔드	MSE 실시간 스트리밍 재생	fetch 스트림 → MediaSource	이수진	2026. 3. 28 9:00	2026-03-30 18:00	0
		프론트엔드	컷편집 기능	In/Out 포인트, 컷 목록 관리	이수진	2026. 3. 28 8:00	2026-03-28 10:00	0
		프론트엔드	전송 진행 현황 UI	진행률 바, 속도, ETA	이수진	2026. 3. 28 10:00	2026. 3. 28 10:30	100
		프론트엔드	전송 이력 페이지	날짜, 파일명, 결과 목록	이수진	2026. 3. 28 11:00	2026. 3. 28 0:00	100
		프론트엔드	설정 페이지	프로필 수정, 에이전트 목록	이수진	2026. 3. 30 17:00	2026. 3. 30 18:00	100
		프론트엔드	이용약관/개인정보처리방침 모달	회원가입 링크 클릭 시 팝업	이수진	2026. 3. 31 17:00	2026. 3. 31 18:00	100
		프론트엔드	스크롤 애니메이션 (fade-up)	랜딩 섹션 진입 시 페이드인	이수진	2026. 4. 2 13:00	2026. 4. 2 14:00	100
	서버 개발	백엔드	비밀번호 변경 API	PUT /auth/password	최하경	2026-03-31 9:00	2026-04-01 18:00	100
		백엔드	인증 미들웨어	Bearer Token 검증, 401 처리	최하경	2026-03-27 9:00	2026-03-28 18:00	100
		백엔드	CORS / Rate Limiting	프론트 origin 허용	최하경	2026. 3. 27	2026-03-28 18:00	0
산출물 정리	문서화	인프라/공통	기획안 작성 및 제출			2026. 3. 26	2026-03-28	100
		인프라/공통	SBOM 보고서 작성			2026. 3. 26	2026-03-28	100
		인프라/공통	발표자료 PPT 제작		최하경	2026. 3. 27	2026. 4. 3	50
		인프라/공통	개발자 매뉴얼 작성(doc)	기능명세서, api 설계, 설계에 대한 예제, Sbom(사용 라이브러리), 테스트 및 상관관계				0
		인프라/공통	사용 설명서 문서화(ppt)	우리 제품은 어떤 기능이 있고 다른 제품과의 차이점 서술				0
		인프라/공통	API 명세서 작성 및 제출			2026. 3. 26	2026. 4. 1	100
	배포?	인프라/공통	서버 구동					0
		인프라/공통	SQLite					0
		인프라/공통	시연					0
		인프라/공통	배포					0
		인프라/공통	유지보수					0
							종합 완료율	55%

최하단에는 AVERAGE함수를 이용하여 종합 완료율 계산, 전체 작업 진전도를 알 수 있게 하였다.

Ⅲ. 시작

3.1 CI/CD 설계와 SBOM, 라이센스 관련

DirectP2P 메인 리포지토리

여러 하위 프로젝트를 `git submodule` 형태로 참조
각 자식 리포지토리에서 `submodule_updated` 이벤트를 수신하면:

- `master` 브랜치에서 submodule을 최신 커밋으로 업데이트
- 정적 분석 및 검증 수행
    - `govulncheck`
    - `go vet`
        i. 이러한 분석 결과를 github issue tracker에 issue로 등록
- Webhook을 통해 Jenkins 호출
    - 전체 프로젝트 빌드
    - 배포 수행

API_Server

GitHub Actions 기반 CI 수행

- push 이벤트 발생 시:
    - 부모(메인) 리포지토리에 `submodule_updated` 이벤트 전송
    - 자동화 테스트 수행

Client_Agent

GitHub Actions 기반 CI 수행

- push 이벤트 발생 시:
    - 부모(메인) 리포지토리에 `submodule_updated` 이벤트 전송
    - 자동화 테스트 수행

Frontend

GitHub Actions 기반 CI 수행
push 이벤트 발생 시:

- 부모 리포지토리에 `submodule_updated` 이벤트 전송
- 정적 리소스 빌드 수행
    - HTML / CSS / JS 번들링

Nginx

Frontend 빌드 산출물(html, css, js)을 대상으로:

- Jenkins를 통해 자동 배포 수행
- Nginx에서 정적 파일 서빙

8.6 전체 흐름 요약

자식 리포지토리(API, Agent, Frontend)에 변경 발생 (push)
각 리포지토리가 부모 리포지토리에 `submodule_updated` 이벤트 전송
메인 리포지토리:

- submodule 최신화
- 정적 분석 수행
- Jenkins Webhook 호출

Jenkins:

- 전체 빌드
- 서비스 및 정적 파일 배포

9.1 기본 식별 정보

- 소프트웨어명 : DIRECT P2P
- 포맷: CycloneDX v1.5 (JSON/XML)
- 작성자/조직 정보 : S-개발자 3조 E둘 I들 (김민수, 배민기, 이수진, 최하경)
- 작성 도구: Manual Compile & syft v1.0.0
- 문서 생성일: 2026년 3월 26일

9.2 대상 소프트웨어 정보

- 버전: v1.0.0
- 공급자: 3조 E둘I들
- 설명: UDP 홀펀칭 기반 NAT 투과 및 QUIC 고속 데이터 전송 서비스
- 패키지 타입: Application (Client-Server Architecture)

9.4 의존성 관계

```
DirectP2P v1.0.0
├── Static Server (Nginx)
│   └── Nginx 1.25.3
├── Web Frontend (JavaScript / React)
│   └── React 18.2.0
├── API Server (Go)
│   ├── Go Runtime 1.22.0
│   ├── Gin v1.9.1
│   ├── SQLite 3.45.1
│   └── libp2p v0.32.0
├── Local Agent (Go)
│   ├── Go Runtime 1.22.0
│   ├── libp2p v0.32.0
│   ├── ffmpeg
│   └── OpenSSL 3.0.13
└── TURN Server (coturn)
    └── coturn 4.6.2
```

9.10 외부 참조

1. 표준 프로토콜 명세 (Standards)

- **RFC 5389:** Session Traversal Utilities for NAT (STUN)
- **RFC 5766:** Traversal Using Relays around NAT (TURN)
- **RFC 9000:** QUIC: A UDP-Based Multiplexed and Secure Transport
- **RFC 5245:** Interactive Connectivity Establishment (ICE)
    - 홀펀칭 절차의 표준입니다.

2. 핵심 라이브러리 및 오픈소스 (Implementations)

- **Gin Framework:** https://github.com/gin-gonic/gin (API 서버 프레임워크)
- **libp2p (Go):** https://github.com/libp2p/go-libp2p (P2P 네트워크 스택)
- **quic-go:** https://github.com/quic-go/quic-go (libp2p 내부 QUIC 전송 계층)
- **go-sqlite3 :** https://github.com/mattn/go-sqlite3 (임베디드 데이터베이스)
- **coturn:** https://github.com/coturn/coturn (TURN 서버 구현체)
- **Nginx :** https://nginx.org/
- **ffmpeg:** https://www.ffmpeg.org
- **jenkins:** https://www.jenkins.io

3. 프로젝트 자원 (Project Assets)

- **GitHub Repository:** https://github.com/E2IDLE/DirectP2P

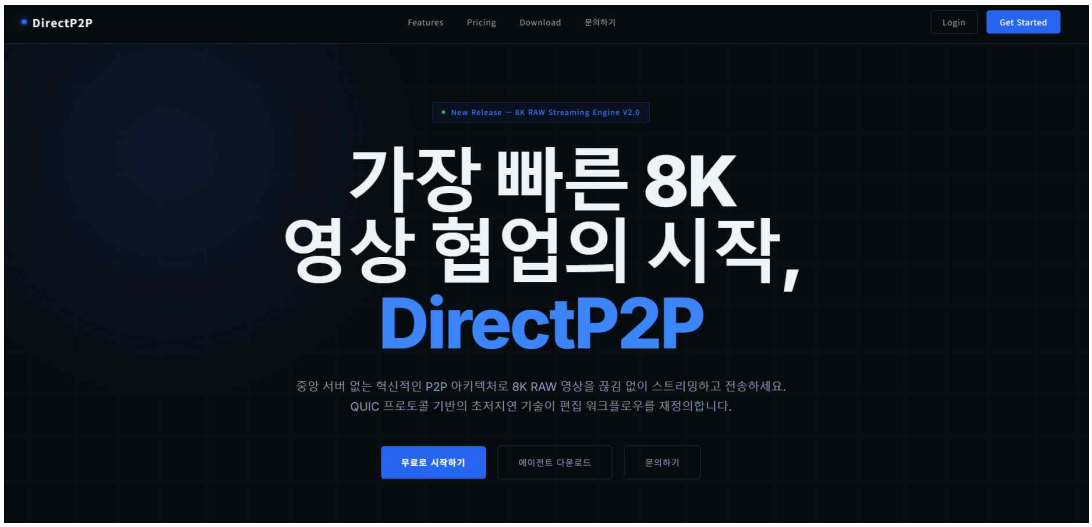
9.11 배포 정보

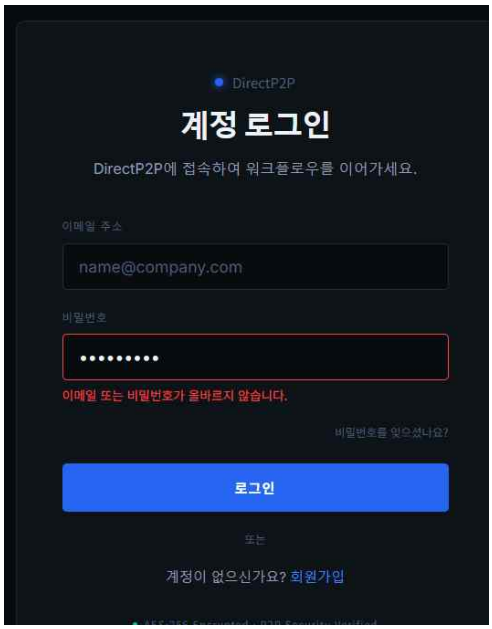
- 본 프로젝트는 GitHub Actions 기반의 CI/CD(지속적 통합 및 배포) 파이프라인을 통해 소프트웨어의 공급망 보안과 배포 무결성을 유지합니다.

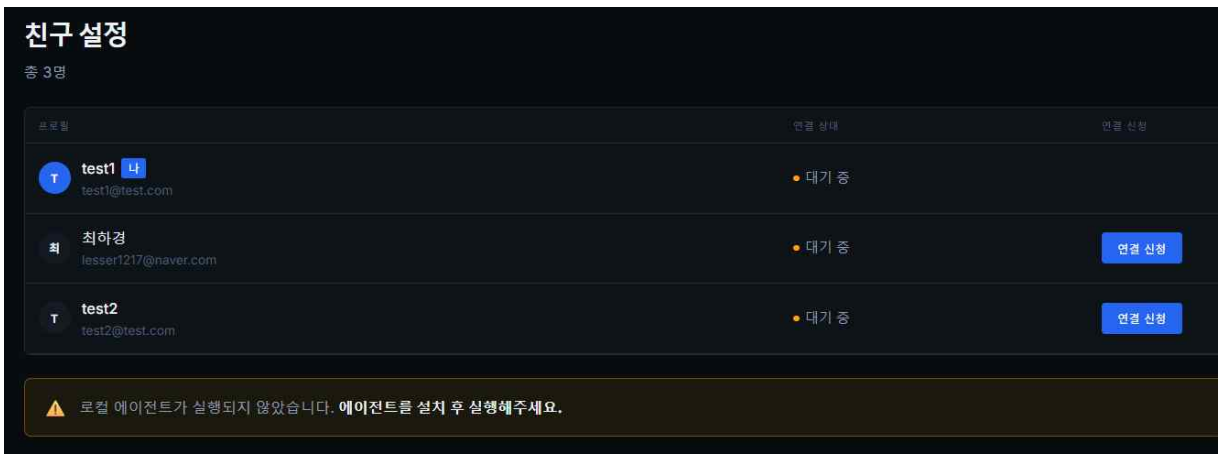
1. 빌드 및 아티팩트 생성 (CI: Continuous Integration)

- **컴파일 및 빌드:** GitHub Actions 워크플로우를 통해 Go 1.22 환경에서 각 OS별(Windows, macOS, Linux) 바이너리를 교차 컴파일(Cross-compile)함.
- **프론트엔드 빌드:** Node.js 환경에서 React 소스를 최적화(Minify/Bundle)하여 정적 에셋 생성.
- **무결성 확보:** 빌드 직후 모든 실행 파일에 대해 SHA-256 체크섬(Checksum)을 생성하여 파일 변조 여부를 추적함

3.3 핵심 기능 동작 예제


DirectP2P
Features
Pricing
Download
문의하기
Login
Get Started
New Release — 8K RAW Streaming Engine V2.0
가장 빠른 8K
영상 협업의 시작,
DirectP2P
중앙 서버 없는 혁신적인 P2P 아키텍처로 8K RAW 영상을 끊김 없이 스트리밍하고 전송하세요.
QUIC 프로토콜 기반의 초저지연 기술이 편집 워크플로우를 재정의합니다.
무료로 시작하기
에이전트 다운로드
문의하기

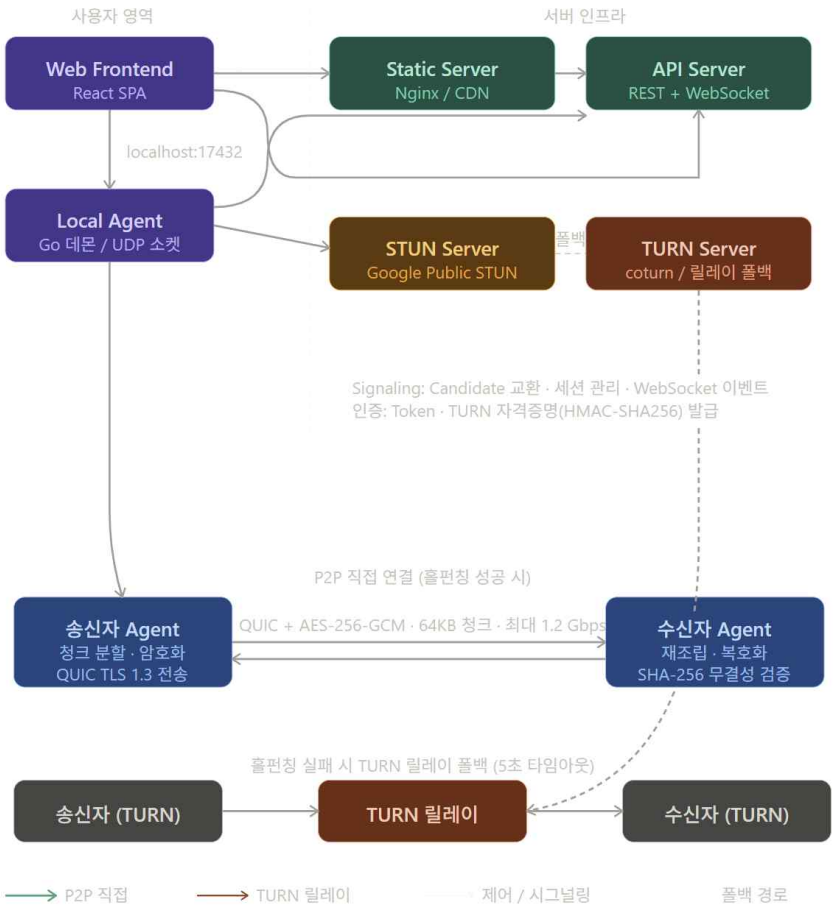

DirectP2P
계정 로그인
DirectP2P에 접속하여 워크플로우를 이어가세요.
이메일 주소
name@company.com
비밀번호
••••••••••
이메일 또는 비밀번호가 올바르지 않습니다.
비밀번호를 잊으셨나요?
로그인
또는
계정이 없으신가요? 회원가입


친구 설정
총 3명
프로필
연결 상태
연결 신청
test1 나
test1@test.com
대기 중
최하경
lesser1217@naver.com
대기 중
연결 신청
test2
test2@test.com
대기 중
연결 신청
로컬 에이전트가 실행되지 않았습니다. 에이전트를 설치 후 실행해주세요.

Ⅳ. 아키텍처 및 개념 설명

4.1 시스템 구조도

DirectP2P는 크게 세 개의 물리적 영역으로 나뉜다.



사용자 영역에는 브라우저에서 실행되는 Web Frontend(React SPA)와 사용자 기기에 설치되는 Local Agent(Go 데몬)가 위치한다. 두 컴포넌트는 localhost:17432의 로컬 HTTP API를 통해 통신하며, 외부 네트워크에는 직접 노출되지 않는다.

서버 인프라 영역은 Static Server(Nginx), API Server(REST + WebSocket), STUN Server, TURN Server로 구성된다. API Server는 사용자 인증, 세션 생성, Candidate 주소 교환(Signaling), WebSocket 실시간 이벤트 전송을 담당하는 시스템의 중추이다. STUN Server는 각

클라이언트의 공인 IP/Port를 파악하는 데 사용되며, TURN Server는 홀펀칭 실패 시 데이터 릴레이 폴백 역할을 한다.

P2P 연결 영역은 홀펀칭 성공 여부에 따라 두 가지 경로로 나뉜다. 홀펀칭이 성공하면 두 Local Agent 간에 QUIC + AES-256-GCM 암호화 기반의 직접 연결이 수립되고(최대 1.2 Gbps), 5초 타임아웃 내에 실패하면 TURN 서버를 통한 릴레이 경로로 자동 전환된다.

4.2 핵심 개념 및 용어 정리

○ **NAT (Network Address Translation) :** 사설 IP를 공인 IP로 변환하는 장치. 가정용 공유기가 대표적 예시로, 내부 기기들은 각자 사설 IP를 갖지만 외부에는 하나의 공인 IP로 노출됩니다. DirectP2P에서 두 피어가 서로 다른 NAT 뒤에 있을 때 직접 연결이 어려워지는 근본 원인이다.

○ **UDP 홀펀칭 :** NAT 장비가 내부에서 외부로 나가는 UDP 패킷을 허용할 때 생성하는 '구멍(매핑 항목)'을 활용하는 기법. 두 피어가 동시에 서로에게 UDP 패킷을 전송하면, 각 NAT가 상대방의 패킷을 허용하는 매핑이 만들어지면서 직접 통신이 가능해진다.

○ **STUN :** 클라이언트가 자신의 NAT 외부 주소(공인 IP:Port)를 파악하기 위해 사용하는 프로토콜이다. Local Agent가 Google Public STUN 서버에 Binding Request를 전송하면, 서버는 요청 패킷을 수신한 공인 주소를 응답으로 돌려준다.

○ **TURN :** UDP 홀펀칭이 실패하는 Symmetric NAT 환경에서 사용하는 릴레이 프로토콜이다. coturn 서버가 두 피어 사이에서 데이터를 중계합니다. 홀펀칭 5초 타임아웃 후 자동 전환되며, 이후 P2P 연결이 복구되면 직접 연결로 업그레이드된다.

○ **QUIC :** UDP 위에서 동작하는 멀티플렉싱 전송 프로토콜입니다. TCP의 신뢰성(재전송, 혼잡 제어)과 TLS 1.3 암호화를 하나의 계층에서 제공한다.

DirectP2P는 P2P 연결이 수립된 직후 그 위에 QUIC 스트림을 생성하여 고속 전송에 활용한다.

○ **Candidate** : 피어가 상대방과 통신하는 데 사용 가능한 IP:Port 주소 목록. Local Agent는 로컬 IP, STUN으로 획득한 공인 IP 등 복수의 Candidate를 수집하여 API Server에 등록하고, 상대방의 Candidate를 조회하여 홀펀칭을 시도한다.

○ **Signaling** : 두 피어가 서로의 Candidate를 교환하고 세션을 수립하는 과정. DirectP2P에서는 API Server의 REST API와 WebSocket이 이 역할을 담당. 피어 간 데이터 자체는 Signaling 서버를 거치지 않는다.

○ **Local Agent** : 사용자 기기에서 실행되는 경량 Go 데몬. UDP 소켓 관리, STUN 요청, 홀펀칭 수행, QUIC 기반 데이터 송수신을 담당하며, localhost:17432의 HTTP API를 통해 Web Frontend와 통신.

○ **AES-256-GCM** : DirectP2P의 모든 P2P 전송 데이터에 적용되는 대칭키 암호화 방식이다. GCM(Galois/Counter Mode) 모드는 암호화와 무결성 검증(인증 태그)을 동시에 제공한다. TURN 릴레이 경로에서도 동일하게 적용되어 서버가 데이터 내용을 볼 수 없다.

○ **HMAC-SHA256** : TURN 서버 접속에 사용되는 임시 자격증명 생성 방식. username은 '만료 Unix Timestamp:사용자ID' 형식이며, password는 HMAC-SHA256(secret_key, username)을 Base64 인코딩한 값. 유효 시간은 발급 후 최대 1시간.

○ **WebSocket** : API Server와 클라이언트(Web Frontend, Local Agent) 간 실시간 양방향 통신 채널. Candidate 등록 알림, 세션 상태 변경, 피어 연결/해제 이벤트 등을 Push 방식으로 전달. 연결 유지는 에이전트의 주기적 ping(agent:ping)으로 확인.

○ **Connection Migration** : QUIC의 Connection ID를 통해, 클라이언트의 IP가 변경(Wi-Fi→LTE 등)되더라도 기존 세션을 끊김 없이 유지하는 기능. DirectP2P는 이를 통해 네트워크 환경 변화에도 P2P 전송이 중단되지 않도록 한다.

4.3 데이터 흐름 다이어그램

전체 End-to-End 흐름은 크게 세 단계로 나뉜다. 시그널링 단계에서 피어 주소를 교환하고, 연결 수립 단계에서 홀펀칭 또는 TURN 폴백으로 경로를 확정한 뒤, 전송 단계에서 실제 데이터를 주고받는다.

Ⅴ. 주요 기능 상세 가이드

5.1. 서비스 소개 및 목적

본 서비스는 UDP 홀펀칭(UDP Hole Punching) 기술을 활용하여 NAT 환경에서도 두 클라이언트가 중앙 서버를 경유하지 않고 직접 P2P 연결을 수립하고, 콘텐츠 데이터(주고 고용량 고화질 영상)를 고속으로 송수신 및 즉석 편집할 수 있는 전송 서비스를 구축하는 것을 목적으로 한다.
사용자 시나리오는 아래의 그림과 같다.

5.2. 코드 예제

① 로그인과 회원가입

auth_handler.go

```
package handler

//auth_handler.go 는 사용자 등록, 로그인, 로그아웃, 계정 삭제 및 비밀번호 변경 기능을 담당하는 핸들러입
니다.

import (
        "API_Server/internal/model"
        "API_Server/internal/service"
        "errors"
        "net/http"

        "github.com/gin-gonic/gin"
)

type AuthHandler struct {
        authSvc *service.AuthService
}
```

```
func NewAuthHandler(authSvc *service.AuthService) *AuthHandler {
	return &AuthHandler{authSvc: authSvc}
}

// POST /auth/register
func (h *AuthHandler) Register(c *gin.Context) {
	var req model.RegisterRequest
	if err := c.ShouldBindJSON(&req); err != nil {
		c.JSON(http.StatusBadRequest, model.ErrorResponse{Code: "BAD_REQUEST", Message:
err.Error()})
		return
	}

	resp, err := h.authSvc.Register(c.Request.Context(), req)
	if err != nil {
		if errors.Is(err, service.ErrEmailExists) {
			c.JSON(http.StatusConflict, model.ErrorResponse{Code: "CONFLICT", Message:
err.Error()})
			return
		}
		c.JSON(http.StatusInternalServerError, model.ErrorResponse{Code: "INTERNAL",
Message: "서버 오류"})
		return
	}

	c.JSON(http.StatusCreated, resp)
}

// POST /auth/login
func (h *AuthHandler) Login(c *gin.Context) {
	var req model.LoginRequest
	if err := c.ShouldBindJSON(&req); err != nil {
		c.JSON(http.StatusBadRequest, model.ErrorResponse{Code: "BAD_REQUEST", Message:
err.Error()})
		return
	}

	resp, err := h.authSvc.Login(c.Request.Context(), req)
	if err != nil {
		if errors.Is(err, service.ErrInvalidLogin) {
			c.JSON(http.StatusUnauthorized, model.ErrorResponse{Code:
"UNAUTHORIZED", Message: err.Error()})
			return
		}
		c.JSON(http.StatusInternalServerError, model.ErrorResponse{Code: "INTERNAL",
Message: "서버 오류"})
		return
	}

	c.JSON(http.StatusOK, resp)
}

// POST /auth/logout 로그아웃
func (h *AuthHandler) Logout(c *gin.Context) {
	token, _ := c.Get("token")
	if err := h.authSvc.Logout(c.Request.Context(), token.(string)); err != nil {
		c.JSON(http.StatusInternalServerError, model.ErrorResponse{Code: "INTERNAL",
Message: "서버 오류"})
		return
	}
	c.Status(http.StatusNoContent)
}

// DELETE /auth/me
func (h *AuthHandler) DeleteAccount(c *gin.Context) {
	userID, _ := c.Get("userID")
	if err := h.authSvc.DeleteAccount(c.Request.Context(), userID.(string)); err != nil {
		c.JSON(http.StatusInternalServerError, model.ErrorResponse{Code: "INTERNAL",
Message: "서버 오류"})
		return
	}
	c.Status(http.StatusNoContent)
}

// PUT /auth/password
func (h *AuthHandler) ChangePassword(c *gin.Context) {
```

```
	var req model.ChangePasswordRequest
	if err := c.ShouldBindJSON(&req); err != nil {
		c.JSON(http.StatusBadRequest, model.ErrorResponse{Code: "BAD_REQUEST", Message:
err.Error()})
		return
	}

	userID, _ := c.Get("userID")
	err := h.authSvc.ChangePassword(c.Request.Context(), userID.(string), req)
	if err != nil {
		if errors.Is(err, service.ErrWrongPassword) {
			c.JSON(http.StatusUnauthorized, model.ErrorResponse{Code:
"UNAUTHORIZED", Message: err.Error()})
			return
		}
		c.JSON(http.StatusInternalServerError, model.ErrorResponse{Code: "INTERNAL",
Message: "서버 오류"})
		return
	}
	c.Status(http.StatusNoContent)
}
```

api.js

```
export const API_BASE = "https://directp2p.happytanuki.kr";
export const AGENT_BASE = "http://127.0.0.1:17432";

export const getToken = () => localStorage.getItem("token");
export const setToken = (t) => localStorage.setItem("token", t);
export const clearToken = () => localStorage.removeItem("token");

let _onUnauth = null;
export const setUnauthHandler = (fn) => { _onUnauth = fn; };

/**
 * fetch wrapper with Bearer token injection.
 * For non-auth endpoints: auto-clears token and calls _onUnauth on 401.
 * Throws { type: "network" } on network failure.
 */
export async function apiFetch(path, options = {}) {
  const token = getToken();
  const isAuthPath = path.startsWith("/auth/");
  const headers = {
    "Content-Type": "application/json",
    ...(token && !isAuthPath ? { Authorization: `Bearer ${token}` } : {}),
    ...(options.headers ?? {}),
  };

  const controller = new AbortController();
  const timeoutId = setTimeout(() => controller.abort(), 15000);

  let res;
  try {
    res = await fetch(`${API_BASE}${path}`, { ...options, headers, signal: controller.signal });
  } catch {
    throw { type: "network" };
  } finally {
    clearTimeout(timeoutId);
  }

  // Auto-logout on 401 for authenticated (non-auth) endpoints
  if (res.status === 401 && !path.startsWith("/auth/") && token) {
    clearToken();
    _onUnauth?.();
  }

  return res;
}

export async function agentCall(method, path, body) {
  const res = await fetch(`${AGENT_BASE}${path}`, {
    method,
    headers: { "Content-Type": "application/json" },
    body: body ? JSON.stringify(body) : undefined,
  });
  return res;
}
```

② Websocket

ws_handler.go

```
package handler

//WebSocket 핸들러는 클라이언트와의 WebSocket 연결을 관리하고, 메시지를 주고받는 역할을 합니다.

import (
	"API_Server/internal/model"
	"API_Server/internal/repository"
	"API_Server/internal/service"
	"API_Server/internal/ws"
	"context"
	"encoding/json"
	"log"
	"net/http"
	"time"

	"github.com/gin-gonic/gin"
	"github.com/gorilla/websocket"
)

var upgrader = websocket.Upgrader{
	ReadBufferSize:  1024,
	WriteBufferSize: 1024,
	CheckOrigin:     func(r *http.Request) bool { return true },
}

type WSHandler struct {
	hub         *ws.Hub
	tokenRepo   *repository.TokenRepository
	sessionSvc  *service.SessionService
}

func NewWSHandler(hub *ws.Hub, tokenRepo *repository.TokenRepository, sessionSvc
*service.SessionService) *WSHandler {
	return &WSHandler{hub: hub, tokenRepo: tokenRepo, sessionSvc: sessionSvc}
}

func (h *WSHandler) HandleWebSocket(c *gin.Context) {
	token := c.Query("token")
	if token == "" {
		c.JSON(http.StatusUnauthorized, model.ErrorResponse{Code: "UNAUTHORIZED",
Message: "토큰이 필요합니다."})
		return
	}

	authToken, err := h.tokenRepo.FindByToken(c.Request.Context(), token)
	if err != nil || authToken == nil {
		c.JSON(http.StatusUnauthorized, model.ErrorResponse{Code: "UNAUTHORIZED",
Message: "유효하지 않은 토큰입니다."})
		return
	}

	conn, err := upgrader.Upgrade(c.Writer, c.Request, nil)
	if err != nil {
		log.Printf("WebSocket 업그레이드 실패: %v", err)
		return
	}

	client := &ws.Client{
		UserID: authToken.UserID,
		Conn:   conn,
		Send:   make(chan []byte, 256),
	}

	h.hub.Register(client)

	go h.writePump(client)
	go h.readPump(client)
}

func (h *WSHandler) writePump(client *ws.Client) {
	defer func() {
		client.Conn.Close()
	}()
```

```
	for {
		msg, ok := <-client.Send
		if !ok {
			client.Conn.WriteMessage(websocket.CloseMessage, []byte{})
			return
		}

		client.Conn.SetWriteDeadline(time.Now().Add(10 * time.Second))
		if err := client.Conn.WriteMessage(websocket.TextMessage, msg); err != nil {
			return
		}
	}
}

func (h *WSHandler) readPump(client *ws.Client) {
	defer func() {
		h.hub.Unregister(client)
		client.Conn.Close()
	}()

	client.Conn.SetReadDeadline(time.Now().Add(60 * time.Second))
	client.Conn.SetPongHandler(func(string) error {
		client.Conn.SetReadDeadline(time.Now().Add(60 * time.Second))
		return nil
	})

	for {
		_, message, err := client.Conn.ReadMessage()
		if err != nil {
			break
		}

		var wsMsg model.WSMessage
		if err := json.Unmarshal(message, &wsMsg); err != nil {
			continue
		}

		switch wsMsg.Event {
		case "agent:ping":
			client.Conn.SetReadDeadline(time.Now().Add(60 * time.Second))
			log.Printf("agent:ping from user %s", client.UserID)

		case "session:status_changed":
			if dataMap, ok := wsMsg.Data.(map[string]interface{}); ok {
				sessionID, _ := dataMap["sessionId"].(string)
				status, _ := dataMap["status"].(string)

				if status == "completed" || status == "error" {
					go h.sessionSvc.UpdateStatus(
						context.Background(), sessionID, status,
					)
				}
			}
		}
	}
}
```

ws/hub.go

```
package ws

// hub.go 는 WebSocket 클라이언트 관리 및 메시지 브로드캐스팅을 담당하는 허브입니다.

import (
	"API_Server/internal/model"
	"encoding/json"
	"log"
	"sync"

	"github.com/gorilla/websocket"
)

// Client 는 하나의 WebSocket 연결을 나타냄
type Client struct {
	UserID string
```

```
        Conn    *websocket.Conn
        Send    chan []byte
}

// Hub 은 모든 WebSocket 클라이언트를 관리
type Hub struct {
        mu          sync.RWMutex
        clients     map[string]*Client // userID → Client
        register    chan *Client
        unregister chan *Client
}

func NewHub() *Hub {
        return &Hub{
                clients:     make(map[string]*Client),
                register:    make(chan *Client),
                unregister: make(chan *Client),
        }
}

func (h *Hub) Run() {
        for {
                select {
                case client := <-h.register:
                        h.mu.Lock()
                        h.clients[client.UserID] = client
                        h.mu.Unlock()
                        log.Printf("WS 클라이언트 등록: %s", client.UserID)

                case client := <-h.unregister:
                        h.mu.Lock()
                        if _, ok := h.clients[client.UserID]; ok {
                                delete(h.clients, client.UserID)
                                close(client.Send)
                        }
                        h.mu.Unlock()
                        log.Printf("WS 클라이언트 해제: %s", client.UserID)
                }
        }
}

func (h *Hub) Register(client *Client) {
        h.register <- client
}

func (h *Hub) Unregister(client *Client) {
        h.unregister <- client
}

// SendToUser 는 특정 사용자에게 메시지를 전송
func (h *Hub) SendToUser(userID string, msg model.WSMessage) {
        h.mu.RLock()
        client, ok := h.clients[userID]
        h.mu.RUnlock()

        if !ok {
                return
        }

        data, err := json.Marshal(msg)
        if err != nil {
                return
        }

        select {
        case client.Send <- data:
        default:
                // 버퍼가 꽉 차면 연결 종료
                h.Unregister(client)
        }
}

// BroadcastToSession 은 특정 세션의 참가자들에게 메시지를 전송
// 간단 구현: targetUserID 로 한 명에게 전송
func (h *Hub) BroadcastToSession(targetUserID string, msg model.WSMessage) {
        h.SendToUser(targetUserID, msg)
}
```

③ QUIC 기반 청크 전송
③ TURN 폴백 자격증명 발급 및 연결
⑤ Websocket

5.3 파라미터/옵션

전체 End-to-End 흐름은 크게 세 단계로 나뉜다. 시그널링 단계에서 피어 주소를 교환하고, 연결 수립 단계에서 홀펀칭 또는 TURN 폴백으로 경로를 확정한 뒤, 전송 단계에서 실제 데이터를 주고받는다.

○ STUN

파라미터	타입	기본값	설명
stun.server	string	stun.l.google.com:	공인 IP 획득에 사용할 STUN 서버 주소. 사내 환경에서는 자체 STUN 서버로 교체 가능
stun.retry_max	int	3	응답 없을 시 재시도 횟수. 초과 시 오류 코드 E1001 반환
stun.timeout	duration	3s	단일 STUN 요청 타임아웃

○ Session

파라미터	타입	기본값	설명
server.port	int	8080	API Server 리슨 포트
server.tls	bool	True	HTTPS 강제 여부. 운영 환경에서는 반드시 true
session.invite_code_ttl	duration	10s	초대 코드 유효 시간. 만료 시 E5002 반환
session.max_candidates	int	8	피어당 등록 가능한 최대 Candidate 수
agent.local_port	int	17432	Local Agent 로컬 HTTP API 포트. localhost 바인딩 전용

5.4 상세 기능 명세

○ 정적 파일 서빙

기능 ID	기능명	설명	우선순위
SS-001	웹 앱 서빙	React/Vue/HTML 기반 SPA 번들 파일 제공 (index.html, JS, CSS, 이미지)	필수
SS-002	HTTPS 지원	TLS/SSL 인증서 적용, HTTP → HTTPS 자동 리다이렉트	필수
SS-003	캐시 제어	정적 자원에 Cache-Control 헤더 설정 (JS/CSS: max-age=31536000, HTML: no-cache)	권장
SS-004	GZIP/Brotli 압축	전송 용량 최소화를 위한 응답 압축 처리	권장
SS-005	SPA 라우팅 처리	모든 경로 요청을 index.html로 fallback (클라이언트 사이드 라우팅 지원)	필수

○ 에이전트 설치 파일 제공

기능 ID	기능명	설명	우선순위
SS-010	플랫폼별 에이전트 배포	Windows(.exe), macOS(.dmg/.pkg), Linux(.deb/.rpm) 설치 파일 다운로드 URL 제공	필수
SS-011	버전 메타데이터 제공	최신 에이전트 버전 정보 JSON API 엔드포인트 (/agent/latest.json)	권장
SS-012	파일 무결성 검증	SHA-256 체크섬 파일(.sha256) 함께 제공하여 다운로드 무결성 보장	권장
SS-013	다운로드 통계	에이전트 다운로드 횟수 및 플랫폼별 통계 수집 (익명)	선택

VI. API 레퍼런스

6.1 엔드포인트 및 함수 목록

○ 인증 활용

메서드	CRUD	엔드포인트	설명	인증	소유권
POST	C	/auth/register	회원가입	X	-
POST	C	/auth/login	로그인/토큰 발급	X	-
POST	U	/auth/logout	로그아웃/토큰 삭제	토큰	본인
DELETE	D	/auth/me	회원 탈퇴	토큰	본인

○ 사용자 관련

메서드	CRUD	엔드포인트	설명	인증	소유권
GET	R	/users/me	내 프로필 조회	토큰	본인
PATCH	U	/users/me	프로필 수정	토큰	본인
GET	R	/users/me/agents	등록 에이전트 목록	토큰	본인

○ CANTDIDATE / TURN

메서드	CRUD	엔드포인트	설명	인증	소유권
POST	C	/sessions/:id/candidates	Candidate 주소 등록	토큰	세션 참여자
GET	R	/sessions/:id/candidates	상대방 Candidate 조회	토큰	세션 참여자
POST	C	/sessions/:id/turn-credentials	TURN 자격증명 발급	토큰	세션 참여자

○ 세션 관련

메서드	CRUD	엔드포인트	설명	인증	소유권
POST	R	/sessions	전송 세션 생성	토큰	본인
POST	U	/sessions/:id/join	세션 참여	토큰	세션 참여자
GET	R	/sessions/:id	세션 상태 조회	토큰	세션 참여자
DELETE	D	/sessions/:id	세션 종료	토큰	세션 참여자
GET	R	/sessions (이력)	과거 세션 이력 조회	토큰	본인

○ AGENT - 상태/세션

메서드	CRUD	엔드포인트	설명	인증	소유권
GET	R	/status	에이전트 상태	불필요	-
GET	R	/session	현재 세션 정보	불필요	-
POST	C	/session/start	세션 시작 요청	불필요	-
POST	C	/session/join	세션 참여 요청	불필요	-

○ AGENT - 전송

메서드	CRUD	엔드포인트	설명	인증	소유권
POST	C	/peers/:id	파일 전송 시작	불필요	-
GET	R	/transfer/ progress	전송 진행률 조회	불필요	-
POST	C	/transfer/cancel	전송 취소	불필요	-

○ AGENT- 피어/로그

메서드	CRUD	엔드포인트	설명	인증	소유권
GET	R	/peers	연결된 피어 목록	불필요	-
GET	R	**/logs**	에이전트 로그 조회	불필요	-

○ Static Server

메서드	CRUD	엔드포인트	설명	인증	소유권
GET	R	/agent/download/:platform	에이전트 설치 파일	불필요	-
GET	R	/agent/latest.json	최신 버전 메타데이터	불필요	-

6.2 요청/응답 형식

모든 HTTP 통신은 Content-Type:application/JSON을 기본으로 사용한다. Websocket 이벤트는 JSON 텍스트 프레임으로 송수신하며, 인증이 필요한 모든 엔드포인트는 Authorization: Bearea〈token〉 헤더가 필요하다.

6.2.1 공통 읍답 구조

성공 및 실패 응답은 아래 공통 Envelope 구조를 따른다.

```
// 성공 응답(HTTP 2xx)
{
  "success": true,
  "data": { ... },          // 실제 반환 데이터(엔드포인트마다 상이)
  "meta": {                   // 선택- 페이지네이션 등 부가 정보
    "page": 1,
    "total": 100
  }
}

// 실패 응답(HTTP 4xx / 5xx)
{
  "success": false,
  "error": {
```

```
    "code": "E5001",          // 에러 코드(6.3절 참조)
    "message": "인증 토큰이 만료되었습니다.",
    "details": { ... }      // 선택- 필드 검증 오류 등 상세 정보
  }
}
```

6.2.2 인증 (Auth)

POST **/auth/register** – 회원가입

Request Body

```
{
  "email":     "user@example.com",    // string, required, 이메일 형식
  "password": "P@ssw0rd!",             // string, required, 8자 이상
  "nickname": "홍길동"                 // string, optional
}
```

Response 201 Created

```
{
  "success": true,
  "data": {
    "userId":    "usr_abc123",
    "email":     "user@example.com",
    "nickname": "홍길동",
    "createdAt": "2026-04-01T12:00:00Z"
  }
```

POST **/auth/login** – 로그인(토큰 발급)

Request Body

```
{
  "email":     "user@example.com",
  "password": "P@ssw0rd!"
}
```

Response 200 OK

```
{
  "success": true,
  "data": {
    "token":      "eyJhbGci...",    // Bearer 토큰
    "expiresAt": "2026-04-08T12:00:00Z"
  }
}
```

POST **/auth/logout** – 로그아웃

Header: Authorization: Bearer ⟨token⟩

Response 200 OK

```
{
  "success": true,
  "data": { "message": "로그아웃 완료" }
```

```
}
```

DELETE **/auth/me** – 회원탈퇴

Header: Authorization: Bearer <token>

Response 200 OK

```
{
  "success": true,
  "data": { "message": "계정이 삭제되었습니다." }
}
```

6.2.3 사용자 / 에이전트 관리 (Users)

GET **/users/me** — 내 프로필 조회

Response 200 OK

```
{
  "success": true,
  "data": {
    "userId":          "usr_abc123",
    "email":           "user@example.com",
    "nickname":        "홍길동",
    "createdAt":       "2026-04-01T12:00:00Z",
    "agentConnected": true      // 로컬 에이전트 연결 여부
  }
}
```

GET **/users/me/agents** – 등록된 에이전트 목록

Response 200 OK

```
{
  "success": true,
  "data": {
    "agents": [
      {
        "agentId":    "agt_xyz789",
        "platform":   "windows",    // windows | macos | linux
        "version":    "1.0.0",
        "lastSeen":   "2026-04-03T09:00:00Z",
        "online":     true
      }
    ]
  }
}
```

6.2.4 세션 관리 (Sessions)

POST **/sessions** – 전송 세션 생성(송신자)

Request Body

```
{
  "agentId": "agt_xyz789",        // 송신에 사용할 에이전트ID
  "fileName": "Scene_01_RAW.mov",
  "fileSize": 8589934592,          // bytes (8 GB)
  "mimeType": "video/quicktime",
  "checksum": "sha256:abcdef..."  // SHA-256 hex
}
```

Response 201 Created

```
{
  "success": true,
  "data": {
    "sessionId": "ses_001",
    "inviteCode": "ABCD-1234",     // 수신자에게 공유할 코드
    "status": "waiting",        // waiting | connecting | connected | done | error
    "createdAt": "2026-04-03T10:00:00Z"
  }
}
```

POST **/sessions/:id/join** – 세션 참여(수신자)

Request Body

```
{
  "inviteCode": "ABCD-1234",
  "agentId": "agt_rxv456"
}
```

Response 200 OK

```
{
  "success": true,
  "data": {
    "sessionId": "ses_001",
    "sender": {
      "nickname": "홍길동",
      "agentId": "agt_xyz789"
    },
    "fileInfo": {
      "fileName": "Scene_01_RAW.mov",
      "fileSize": 8589934592,
      "mimeType": "video/quicktime"
    },
    "status": "connecting"
  }
}
```

GET **/sessions/:id** – 세션 상태 조회

Response 200 OK

```
{
  "success": true,
  "data": {
```

```
      "sessionId":      "ses_001",
      "status":         "connected",
      "connectionType": "p2p",        // p2p | turn
      "rttMs":          14,
      "startedAt":      "2026-04-03T10:01:00Z"
    }
  }
```

DELETE **/sessions/:id** – 세션 종료

Response 200 OK

```
  {
    "success": true,
    "data": { "message": "세션이 종료되었습니다." }
  }
```

6.2.5 Candidate 주소 교환

POST **/sessions/:id/candidates** – Candidate 주소 등록

Request Body

```
  {
    "candidates": [
      {
        "type":      "host",            // host | srflx(STUN) | relay(TURN)
        "ip":        "192.168.0.10",
        "port":      54321,
        "protocol":  "udp"
      },
      {
        "type":  "srflx",
        "ip":    "203.0.113.5",
        "port":  54321,
        "protocol": "udp"
      }
    ]
  }
```

Response 200 OK

```
  {
    "success": true,
    "data": { "registered": 2 }
  }
```

GET **/sessions/:id/candidates** – 상대방Candidate 조회

Response 200 OK

```
  {
    "success": true,
    "data": {
      "peerId": "usr_peer99",
```

```
      "candidates": [
        { "type": "host",  "ip": "10.0.0.2",     "port": 49152, "protocol": "udp" },
        { "type": "srflx", "ip": "198.51.100.7","port": 49152, "protocol": "udp" }
      ]
    }
  }
```

POST **/sessions/:id/turn-credentials** – TURN 자격증명 발급

Response 200 OK

```
  {
    "success": true,
    "data": {
      "turnServer": "turn.directp2p.io:3478",
      "username":   "1714780800:usr_abc123",    // 만료Unix timestamp:userId
      "password":   "Base64(HMAC-SHA256(secret, username))",
      "expiresAt":  "2026-04-03T11:00:00Z"      // 발급 후1시간
    }
  }
```

6.2.6 WebSocket 이벤트(WS /ws)

연결 방법: ws(s)://⟨host⟩/ws | Header: Authorization: Bearer ⟨token⟩

모든 이벤트는 아래 공통JSON 구조를 사용

```
  {
    "event":   "session:peer_joined",  // 이벤트명
    "payload": { ... },                // 이벤트별 데이터
    "ts":      "2026-04-03T10:01:00Z"  // 서버 발생 시각
  }
```

이벤트명	방향	페이로드 주요 필드	설명
session:peer_joined	서버→클	sessionId, peerId, nickname	상대방 세션 참여 알림
session:candidate_added	서버→클	sessionId, candidate{}	새Candidate 주소 등록됨
session:status_changed	서버→클	sessionId, status, connectionType	세션 상태 변경(connecting→connected 등)
session:peer_disconnected	서버→클	sessionId, peerId	상대방 연결 끊김
agent:ping	클→서버	{} (빈payload)	에이전트 생존Heartbeat (30초 간격)

session:peer_joined 예시

```
  {
    "event": "session:peer_joined",
    "payload": {
      "sessionId": "ses_001",
      "peerId":    "usr_peer99",
      "nickname":  "수신자A"
```

```
    },
    "ts": "2026-04-03T10:01:00Z"
  }
```

6.2.7 로컬 에이전트HTTP API (localhost:17432)

에이전트가 로컬에서 구동하는HTTP 서버입니다. 웹 프론트엔드가 폴링 또는 요청을 통해 통신한다. 외부 인터넷에서 접근할 수 없으며localhost 바인딩만 허용한다.

GET **/status** – 에이전트 상태 조회

Response 200 OK

```
  {
    "success": true,
    "data": {
      "version":    "1.0.0",
      "online":     true,
      "userId":     "usr_abc123",
      "platform":   "windows",
      "localPort":  54321
    }
  }
```

POST **/session/start** – 전송 세션 시작

Request Body

```
  {
    "sessionId": "ses_001",
    "filePath":  "C:₩₩Videos₩₩Scene_01_RAW.mov"
  }
```

Response 200 OK

```
  {
    "success": true,
    "data": { "message": "전송 세션이 시작되었습니다." }
  }
```

POST **/session/join** – 세션 참여(수신측)

Request Body

```
  {
    "sessionId": "ses_001",
    "savePath":  "D:₩₩Downloads"
  }
```

Response 200 OK

```
  {
    "success": true,
    "data": { "message": "세션에 참여했습니다." }
  }
```

GET **/transfer/progress** – 전송 진행률 조회

Response 200 OK

```
{
  "success": true,
  "data": {
    "sessionId":          "ses_001",
    "fileName":           "Scene_01_RAW.mov",
    "totalBytes":         8589934592,
    "transferredBytes":4294967296,
    "progressPct":        50.0,
    "speedBps":           157286400,    // ~150 MB/s
    "etaSeconds":         27,
    "status":             "transferring" // transferring | paused | done | error
  }
}
```

POST **/peers/:id** – 지정 피어에 데이터 전송 요청

Request Body

```
{
  "filePath":  "C:₩₩Videos₩₩Scene_01_RAW.mov",
  "sessionId": "ses_001"
}
```

Response 200 OK

```
{
  "success": true,
  "data": { "transferId": "txf_001", "message": "전송이 시작되었습니다." }
}
```

POST **/transfer/cancel** – 전송 취소

Request Body

```
{
  "sessionId": "ses_001"
}
```

Response 200 OK

```
{
  "success": true,
  "data": { "message": "전송이 취소되었습니다." }
}
```

6.2.8 P2P 청크 전송 프로토콜(UDP / QUIC)

P2P 연결 수립 후 에이전트끼리 직접 교환하는 바이너리 프레임 구조다. QUIC 스트림 위에서 동작하며 아래 헤더를 앞에 붙인다.

magic	4 bytes	uint32	0x44503250 ("DP2P") – 프레임 식별자
version	1 byte	uint8	프로토콜 버전(현재0x01)
type	1 byte	uint8	0x01=META 0x02=CHUNK 0x03=ACK 0x04=NACK 0x05=FIN
seq	8 bytes	uint64	청크 시퀀스 번호(0부터 증가)

payloadLen	4 bytes	uint32	페이로드 길이(bytes)
payload	N bytes	bytes	실제 데이터(AES-256-GCM 암호화 적용)
checksum	4 bytes	uint32	CRC32 – 헤더+페이로드 전체 체크섬

META 페이로드(type=0x01) – 전송 시작 전 파일 정보 전달

```
{
  "fileName":    "Scene_01_RAW.mov",
  "fileSize":    8589934592,
  "mimeType":    "video/quicktime",
  "sha256":      "abcdef...",          // 전체 파일SHA-256
  "chunkSize":   65536,                  // 청크 크기(기본64 KB)
  "totalChunks": 131072
}
```

VII. 에러 처리와 디버깅

7.1 주요 에러코드 및 의미

에러 응답은6.2.1에서 정의한 공통Envelope의error.code 필드를 통해 반환된다. 코드는 영역별로 E1xxx~E6xxx로 분류하며, 각 코드에 대한HTTP 상태 코드, 설명, 클라이언트 처리 방안을 함께 정의한다.

7.1.1 E1xxx – STUN 관련

코드	HTTP	메시지	원인	처리 방안
E1001	503	STUN 서버 응답 없음	STUN 서버 타임아웃 또는 네트워크 단절	최대3회 재시도 후 에러 보고; 사용자에게 네트워크 확인 안내
E1002	502	공인 주소 획득 실패	STUN 응답이 왔으나 Mapped Address 없음	네트워크 설정 확인 안내; TURN 폴백 전환 시도

E1001 응답 예시

```
{
  "success": false,
  "error": {
    "code":    "E1001",
    "message": "STUN 서버에서 응답을 받지 못했습니다.",
    "details": {
      "stunServer": "stun.l.google.com:19302",
      "retries":    3
    }
  }
}
```

7.1.2 E2xxx – 홀펀칭 관련

코드	HTTP	메시지	원인	처리 방안
E2001	408	홀펀칭 타임아웃	5초 이내 양방향UDP 패킷 미수신	TURN 릴레이로 자동 폴백 전환
E2002	503	모든Candidate 연결 실패	모든 후보 주소에서 연결 수립 불가	TURN 릴레이로 자동 폴백 전환

E2001 응답 예시(에이전트→ 웹 프론트엔드)

```
{
  "success": false,
  "error": {
    "code":      "E2001",
    "message": "홀펀칭이5초 내에 완료되지 않았습니다.",
    "details": {
      "triedCandidates": 3,
      "fallback":            "turn"
    }
  }
}
```

7.1.3 E3xxx – TURN 관련

코드	HTTP	메시지	원인	처리 방안
E3001	401	TURN 인증 실패	HMAC 자격증명 불일치 또는 만료	API Server에서 자격증명 재발급 후 재시도
E3002	503	TURN 서버 연결 불가	TURN 서버 장애 또는 네트워크 문제	최대3회 재시도; 실패 시 사용자 오류 안내

E3001 응답 예시

```
{
  "success": false,
  "error": {
    "code":      "E3001",
    "message": "TURN 인증에 실패했습니다. 자격증명이 만료되었을 수 있습니다.",
    "details": {
      "turnServer": "turn.directp2p.io:3478",
      "hint":          "POST /sessions/:id/turn-credentials 로 자격증명을 재발급받으세요."
    }
  }
}
```

7.1.4 E4xxx – 데이터 전송 관련

코드	HTTP	메시지	원인	처리 방안
E4001	408	청크 전송 타임아웃	지정 시간 내ACK 미수신	NACK으로 해당 청크 재전송 요청
E4002	422	무결성 검증 실패	수신 데이터SHA-256 불일치	전체 파일 재전송 요청
E4003	507	디스크 용량 부족	수신측 저장 공간 부족	수신측 저장 공간 확보 안내 후 재시도
E4004	400	지원되지 않는 파일 형식	MIME 타입 허용 목록 외 파일	허용 파일 형식 안내
E4005	413	파일 크기 초과	단일 전송 파일 크기 제한 초과(기본: 100 GB)	파일 분할 전송 권장

E4002 응답 예시

```
{
  "success": false,
  "error": {
    "code":      "E4002",
    "message": "수신한 파일의 무결성 검증에 실패했습니다.",
    "details": {
      "expected": "sha256:abcdef...",
      "actual":    "sha256:123456...",
      "action":    "retransmit"
    }
  }
}
```

7.1.5 E5xxx – API 서버 관련

코드	HTTP	메시지	원인	처리 방안
E5001	401	인증 토큰 만료	Bearer 토큰 유효기간 초과	로그인 재요청 후 새 토큰 발급
E5002	404	세션 없음	존재하지 않거나 이미 종료된 세션ID	세션 재생성 안내
E5003	422	요청 파라미터 오류	필수 필드 누락 또는 유효성 검사 실패	details 필드의 필드별 오류 내용 참고
E5004	409	이미 참여된 세션	동일 세션에 이미 수신자가 참여한 상태	새 세션 생성 안내
E5005	429	Rate Limit 초과	단위 시간 내 요청 횟수 초과(100 req/min 기본)	Retry-After 헤더 참고 후 재시도
E5006	500	내부 서버 오류	서버 내부 예외 발생	잠시 후 재시도; 반복 시 팀에 문의

E5003 응답 예시(필드 유효성 오류)

```
{
  "success": false,
  "error": {
```

```
    "code":    "E5003",
    "message": "요청 파라미터가 올바르지 않습니다.",
    "details": {
      "fields": {
        "email":    "유효한 이메일 형식이 아닙니다.",
        "password": "비밀번호는8자 이상이어야 합니다."
      }
    }
  }
}
```

E5005 Rate Limit 응답 예시

```
// HTTP Response Header
HTTP/1.1 429 Too Many Requests
Retry-After: 30

// Body
{
  "success": false,
  "error": {
    "code":    "E5005",
    "message": "요청 횟수 제한을 초과했습니다. 30초 후 재시도하세요.",
    "details": {
      "retryAfterSeconds": 30,
      "limit":             100,
      "window":            "1m"
    }
  }
}
```

7.1.6 E6xxx – 로컬 에이전트 관련

코드	설명	원인	처리방안
E6001	에이전트 미실행	로컬 에이전트 프로세스가 구동되지 않음	에이전트 설치/실행 안내 팝업 표시
E6002	에이전트 버전 불일치	에이전트 버전이 서버 최소 요구 버전 미만	자동 업데이트 안내 또는 수동 업데이트 링크 제공
E6003	UDP 포트 바인딩 실패	로컬 방화벽 또는 포트 충돌	방화벽 설정 확인 안내; 대체 포트 자동 시도
E6004	파일 접근 불가	지정한 파일 경로가 없거나 권한 부족	파일 경로 및 권한 확인 안내

E6001 응답 예시(에이전트→ 웹 프론트엔드)

```
{
  "success": false,
  "error": {
    "code":    "E6001",
    "message": "DirectP2P 에이전트가 실행되어 있지 않습니다.",
```

```
        "details": {
          "downloadUrl": "https://directp2p.io/agent/download/windows",
          "hint":        "에이전트를 설치하고 실행한 뒤 새로고침하세요."
        }
      }
    }
```

7.1.7 에러 코드 전체 요약

코드	영역	메시지	HTTP	처리방안
E1001	STUN	STUN 서버 응답 없음	503	3회 재시도 후 에러 보고
E1002	STUN	공인 주소 획득 실패	502	네트워크 설정 확인 안내
E2001	홀펀칭	홀펀칭 타임아웃	408	TURN 자동 폴백
E2002	홀펀칭	모든Candidate 연결 실패	503	TURN 자동 폴백
E3001	TURN	TURN 인증 실패	401	자격증명 재발급 후 재시도
E3002	TURN	TURN 서버 연결 불가	503	3회 재시도 후 오류 안내
E4001	전송	청크 전송 타임아웃	408	NACK 재전송 요청
E4002	전송	무결성 검증 실패	422	전체 재전송 요청
E4003	전송	디스크 용량 부족	507	저장 공간 확보 안내
E4004	전송	지원되지 않는 파일 형식	400	허용 형식 안내
E4005	전송	파일 크기 초과	413	파일 분할 권장
E5001	API	인증 토큰 만료	401	재로그인 후 토큰 갱신
E5002	API	세션 없음	404	세션 재생성 안내
E5003	API	요청 파라미터 오류	422	필드별 오류 안내
E5004	API	이미 참여된 세션	409	새 세션 생성 안내
E5005	API	Rate Limit 초과	429	Retry-After 후 재시도
E5006	API	내부 서버 오류	500	잠시 후 재시도
E6001	에이전트	에이전트 미실행	-	설치/실행 안내 팝업
E6002	에이전트	에이전트 버전 불일치	-	업데이트 안내
E6003	에이전트	UDP 포트 바인딩 실패	-	방화벽/포트 설정 확인
E6004	에이전트	파일 접근 불가	-	경로/권한 확인 안내

7.2 디버깅 방법

컴포넌트별 디버깅 진입점과 주요 확인 포인트를 정리하였다. 각 컴포넌트는 독립 프로세스이므로 해당 컴포넌트의 로그와 상태를 개별적으로 확인해야 한다.

7.2.1 API 서버 디버깅

환경변수 설정(디버그 모드 활성화)

```
# configs/.env
GIN_MODE=debug            # release → debug로 변경
LOG_LEVEL=debug           # info | debug | warn | error
LOG_FORMAT=text           # text (개발) | json (운영)
```

OnPush.yaml 작성하여 자동 빌드

```
name: OnPush

on:
  push:
    branches: [ master, release ] #master, realease 에만 (merge하기 전에)

jobs:
  notify-parent:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Generate DIRECTP2P_SECRET_KEY.pem key file
        run: echo "${{ secrets.DIRECTP2P_SECRET_KEY }}" > DIRECTP2P_SECRET_KEY.pem
      - name: Generate Github JWT
        # This script will set GITHUB_JWT environment variable
        # APP_ID=3210791
        run: |
          chmod +x ./.github/scripts/GenerateGithubJWT.sh
          ./.github/scripts/GenerateGithubJWT.sh 3210791 DIRECTP2P_SECRET_KEY.pem
        shell: bash
      - name: Get organization installation
        id: install_raw
        run: |
          RESPONSE=$(curl -s -L ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/orgs/E2IDLE/installation)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract installation id
        id: install
        run: echo "id=${{ fromJson(steps.install_raw.outputs.json).id }}" >> $GITHUB_OUTPUT
      - name: Get installation token
        id: token_raw
        run: |
          RESPONSE=$(curl -s -X POST ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/app/installations/${{ steps.install.outputs.id }}/access_tokens)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract token
        id: token
        run: echo "INSTALLATION_TOKEN=${{ fromJson(steps.token_raw.outputs.json).token }}" >>
$GITHUB_ENV
      - name: Trigger parent repo
        run: |
          curl -X POST ₩
            -H "Authorization: Bearer $INSTALLATION_TOKEN" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/repos/E2IDLE/DirectP2P/dispatches ₩
            -d '{
            "event_type":"submodule_updated",
              "client_payload": {
                "branch": "${{ github.ref_name }}"
              }
            }'
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Build
        run: go build -o build/ cmd/server/main.go #자동으로 빌드되도록 사용해보기


#test : githubactions
#build : jenkins
```

GenerateGithubJWT.sh 작성

```
#!/usr/bin/env bash

client_id=$1 # Client ID as first argument

pem=$( cat $2 ) # file path of the private key as second argument

now=$(date +%s)
iat=$((${now} - 60)) # Issues 60 seconds in the past
exp=$((${now} + 600)) # Expires 10 minutes in the future

b64enc() { openssl base64 | tr -d '=' | tr '/+' '_-' | tr -d '₩n'; }

header_json='{
    "typ":"JWT",
    "alg":"RS256"
}'
# Header encode
header=$( echo -n "${header_json}" | b64enc )

payload_json="{
    ₩"iat₩":${iat},
    ₩"exp₩":${exp},
    ₩"iss₩":₩"${client_id}₩"
}"
# Payload encode
payload=$( echo -n "${payload_json}" | b64enc )

# Signature
header_payload="${header}"."${payload}"
signature=$(
    openssl dgst -sha256 -sign <(echo -n "${pem}") ₩
    <(echo -n "${header_payload}") | b64enc
)

# Create JWT
JWT="${header_payload}"."${signature}"

echo "GITHUB_JWT=$JWT" >> $GITHUB_ENV
```

서버 직접 실행 및 로그 확인

```
# 바이너리 직접 실행
cd cmd/api-server
go run main.go

# 또는 빌드 후 실행
go build -o api-server ./cmd/api-server
./api-server
```

엔드포인트 상태 확인(curl)

```
# 헬스체크
curl -v http://localhost:8080/health

# 로그인 테스트
curl -X POST http://localhost:8080/auth/login ₩
  -H "Content-Type: application/json" ₩
  -d '{"email":"test@test.com","password":"P@ssw0rd!"}'

# 인증 필요 엔드포인트(TOKEN 교체 필요)
curl -H "Authorization: Bearer <TOKEN>" ₩
  http://localhost:8080/users/me
```

WebSocket 연결 확인

```
# websocat 설치 후 테스트(https://github.com/vi/websocat)
```

```
websocat -H "Authorization: Bearer <TOKEN>" ₩
  ws://localhost:8080/ws

# 서버로ping 전송
{"event":"agent:ping","payload":{}}
```

데이터베이스 직접 조회(SQLite)

```
# SQLite CLI로 직접 확인
sqlite3 ./data/directp2p.db

-- 세션 상태 조회
SELECT id, status, created_at FROM sessions ORDER BY created_at DESC LIMIT 10;

-- 유저 목록
SELECT id, email, created_at FROM users;

-- 종료되지 않은 세션 확인
SELECT * FROM sessions WHERE status NOT IN ("done","error");
```

7.2.2 에이전트 디버깅

에이전트 상태 폴링(웹 브라우저/ curl)

```
# 에이전트 상태 확인
curl http://localhost:17432/status

# 현재 세션 정보
curl http://localhost:17432/session

# 전송 진행률
curl http://localhost:17432/transfer/progress

# 연결된 피어 목록
curl http://localhost:17432/peers

# 에이전트 로그(최근50줄)
curl http://localhost:17432/logs
```

에이전트 수동 실행(디버그 플래그)

```
# Windows
.₩agent.exe --log-level=debug --port=17432

# macOS / Linux
./agent.bin --log-level=debug --port=17432

# 로그를 파일로 저장하면서 실행
./agent.bin --log-level=debug 2>&1 | tee agent_debug.log
```

OnMasterReleasePush.yaml

```
name: OnMasterReleasePush

on:
  push:
    branches: [ master, release ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
```

```
      - name: Build
        run: go build -o build/ cmd/client/main.go
  notify-parent:
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Generate DIRECTP2P_SECRET_KEY pem key file
        run: echo "${{ secrets.DIRECTP2P_SECRET_KEY }}" > DIRECTP2P_SECRET_KEY.pem
      - name: Generate Github JWT
        # This script will set GITHUB_JWT environment variable
        # APP_ID=3210791
        run: |
          chmod +x ./.github/scripts/GenerateGithubJWT.sh
          ./.github/scripts/GenerateGithubJWT.sh 3210791 DIRECTP2P_SECRET_KEY.pem
        shell: bash
      - name: Get organization installation
        id: install_raw
        run: |
          RESPONSE=$(curl -s -L ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/orgs/E2IDLE/installation)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract installation id
        id: install
        run: echo "id=${{ fromJson(steps.install_raw.outputs.json).id }}" >> $GITHUB_OUTPUT
      - name: Get installation token
        id: token_raw
        run: |
          RESPONSE=$(curl -s -X POST ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/app/installations/${{ steps.install.outputs.id }}/access_tokens)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract token
        id: token
        run: echo "INSTALLATION_TOKEN=${{ fromJson(steps.token_raw.outputs.json).token }}" >>
$GITHUB_ENV
      - name: Trigger parent repo
        run: |
          curl -X POST ₩
            -H "Authorization: Bearer $INSTALLATION_TOKEN" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/repos/E2IDLE/DirectP2P/dispatches ₩
            -d '{
            "event_type":"submodule_updated",
              "client_payload": {
                "branch": "${{ github.ref_name }}"
              }
            }'
```

Test.yml

```
name: PerformTest

on:
  push:
    branches: [ master, develop, release ]

jobs:
  tests:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Build
        run: go build -o build/ cmd/client/main.go
      - name: Run binary
```

```
          run: build/main &
        - name: Wait for server
          run: |
              for i in {1..10}; do
              if curl -s http://localhost:17432/status > /dev/null; then
                  echo "Server is up"
                  exit 0
              fi
              echo "Waiting..."
              sleep 1
              done
              echo "The server is not showing up.."
              exit 1

        # Tests are able to performed at this point
        - name: /status GET test
          run: |
              STATUS=$(curl -o /dev/null -s -w "%{http_code}₩n" http://localhost:17432/status)

              if [ "$STATUS" -ne 200 ]; then
                  echo "에러 발생: $STATUS"
                  exit 1
              fi
        - name: /status PATCH test
          run: |
              STATUS=$(curl --location --request PATCH 'http://localhost:17432/status' --header
'Content-Type: application/json' --data '{"name":"name"}' -o /dev/null -s -w "%{http_code}₩n")

              if [ "$STATUS" -ne 200 ]; then
                  echo "에러 발생: $STATUS"
                  exit 1
              fi
```

GenerateGithubJWT.sh

```
#!/usr/bin/env bash

client_id=$1 # Client ID as first argument

pem=$( cat $2 ) # file path of the private key as second argument

now=$(date +%s)
iat=$((${now} - 60)) # Issues 60 seconds in the past
exp=$((${now} + 600)) # Expires 10 minutes in the future

b64enc() { openssl base64 | tr -d '=' | tr '/+' '_-' | tr -d '₩n'; }

header_json='{
    "typ":"JWT",
    "alg":"RS256"
}'
# Header encode
header=$( echo -n "${header_json}" | b64enc )

payload_json="{
    ₩"iat₩":${iat},
    ₩"exp₩":${exp},
    ₩"iss₩":₩"${client_id}₩"
}"
# Payload encode
payload=$( echo -n "${payload_json}" | b64enc )

# Signature
header_payload="${header}"."${payload}"
signature=$(
    openssl dgst -sha256 -sign <(echo -n "${pem}") ₩
    <(echo -n "${header_payload}") | b64enc
)
```

```
# Create JWT
JWT="${header_payload}"."${signature}"

echo "GITHUB_JWT=$JWT" >> $GITHUB_ENV
```

7.2.3 NginX

OnMasterReleasePush.yml

```
name: OnMasterReleasePush

on:
  push:
    branches: [ master, release ]

jobs:
  notify-parent:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Generate DIRECTP2P_SECRET_KEY pem key file
        run: echo "${{ secrets.DIRECTP2P_SECRET_KEY }}" >
DIRECTP2P_SECRET_KEY.pem
      - name: Generate Github JWT
        # This script will set GITHUB_JWT environment variable
        # APP_ID=3210791
        run: |
          chmod +x ./.github/scripts/GenerateGithubJWT.sh
          ./.github/scripts/GenerateGithubJWT.sh 3210791
DIRECTP2P_SECRET_KEY.pem
        shell: bash
      - name: Get organization installation
        id: install_raw
        run: |
          RESPONSE=$(curl -s -L \
            -H "Accept: application/vnd.github+json" \
            -H "Authorization: Bearer $GITHUB_JWT" \
            -H "X-GitHub-Api-Version: 2026-03-10" \
            https://api.github.com/orgs/E2IDLE/installation)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract installation id
        id: install
        run: echo "id=${{ fromJson(steps.install_raw.outputs.json).id }}" >>
$GITHUB_OUTPUT
      - name: Get installation token
        id: token_raw
        run: |
          RESPONSE=$(curl -s -X POST \
            -H "Authorization: Bearer $GITHUB_JWT" \
            -H "Accept: application/vnd.github+json" \
            -H "X-GitHub-Api-Version: 2026-03-10" \
            https://api.github.com/app/installations/${{ steps.install.outputs.id
}}/access_tokens)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
```

```
      - name: Extract token
        id: token
        run: echo "INSTALLATION_TOKEN=${{
fromJson(steps.token_raw.outputs.json).token }}" >> $GITHUB_ENV
      - name: Trigger parent repo
        run: |
          curl -X POST \
            -H "Authorization: Bearer $INSTALLATION_TOKEN" \
            -H "Accept: application/vnd.github+json" \
            -H "X-GitHub-Api-Version: 2026-03-10" \
            https://api.github.com/repos/E2IDLE/DirectP2P/dispatches \
            -d '{
            "event_type":"submodule_updated",
              "client_payload": {
                "branch": "${{ github.ref_name }}"
              }
            }'
```

GenerateGithubJWT.sh

```
#!/usr/bin/env bash

client_id=$1 # Client ID as first argument

pem=$( cat $2 ) # file path of the private key as second argument

now=$(date +%s)
iat=$((${now} - 60)) # Issues 60 seconds in the past
exp=$((${now} + 600)) # Expires 10 minutes in the future

b64enc() { openssl base64 | tr -d '=' | tr '/+' '_-' | tr -d '\n'; }

header_json='{
    "typ":"JWT",
    "alg":"RS256"
}'
# Header encode
header=$( echo -n "${header_json}" | b64enc )

payload_json="{
    \"iat\":${iat},
    \"exp\":${exp},
    \"iss\":\"${client_id}\"
}"
# Payload encode
payload=$( echo -n "${payload_json}" | b64enc )

# Signature
header_payload="${header}"."${payload}"
signature=$(
    openssl dgst -sha256 -sign <(echo -n "${pem}") \
    <(echo -n "${header_payload}") | b64enc
)

# Create JWT
JWT="${header_payload}"."${signature}"

echo "GITHUB_JWT=$JWT" >> $GITHUB_ENV
```

docker-compose.yml

```
services:
```

```
  server:
    privileged: true
    build:
      context: .
      dockerfile: server/Dockerfile
    expose:
      - "80"
    volumes:
      - ./data:/app/data
  nginx:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8088:80"
    depends_on:
      - server
```

7.2.4 웹 프론트엔드 디버깅

GenerateGithubJWT.sh

```
#!/usr/bin/env bash

client_id=$1 # Client ID as first argument

pem=$( cat $2 ) # file path of the private key as second argument

now=$(date +%s)
iat=$((${now} - 60)) # Issues 60 seconds in the past
exp=$((${now} + 600)) # Expires 10 minutes in the future

b64enc() { openssl base64 | tr -d '=' | tr '/+' '_-' | tr -d '₩n'; }

header_json='{
    "typ":"JWT",
    "alg":"RS256"
}'
# Header encode
header=$( echo -n "${header_json}" | b64enc )

payload_json="{
    ₩"iat₩":${iat},
    ₩"exp₩":${exp},
    ₩"iss₩":₩"${client_id}₩"
}"
# Payload encode
payload=$( echo -n "${payload_json}" | b64enc )

# Signature
header_payload="${header}"."${payload}"
signature=$(
    openssl dgst -sha256 -sign <(echo -n "${pem}") ₩
    <(echo -n "${header_payload}") | b64enc
)

# Create JWT
JWT="${header_payload}"."${signature}"

echo "GITHUB_JWT=$JWT" >> $GITHUB_ENV
```

OnPush.yml

```
name: OnPush

on:
  push:
    branches: [ master, release ]

jobs:
  notify-parent:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v6
      - name: Generate DIRECTP2P_SECRET_KEY pem key file
        run: echo "${{ secrets.DIRECTP2P_SECRET_KEY }}" >
DIRECTP2P_SECRET_KEY.pem
      - name: Generate Github JWT
        # This script will set GITHUB_JWT environment variable
        # APP_ID=3210791
        run: |
          chmod +x ./.github/scripts/GenerateGithubJWT.sh
          ./.github/scripts/GenerateGithubJWT.sh 3210791 DIRECTP2P_SECRET_KEY.pem
        shell: bash
      - name: Get organization installation
        id: install_raw
        run: |
          RESPONSE=$(curl -s -L ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/orgs/E2IDLE/installation)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract installation id
        id: install
        run: echo "id=${{ fromJson(steps.install_raw.outputs.json).id }}" >>
$GITHUB_OUTPUT
      - name: Get installation token
        id: token_raw
        run: |
          RESPONSE=$(curl -s -X POST ₩
            -H "Authorization: Bearer $GITHUB_JWT" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/app/installations/${{ steps.install.outputs.id
}}/access_tokens)

          echo "json="$RESPONSE"" >> $GITHUB_OUTPUT
      - name: Extract token
        id: token
        run: echo "INSTALLATION_TOKEN=${{
fromJson(steps.token_raw.outputs.json).token }}" >> $GITHUB_ENV
      - name: Trigger parent repo
        run: |
          curl -X POST ₩
            -H "Authorization: Bearer $INSTALLATION_TOKEN" ₩
            -H "Accept: application/vnd.github+json" ₩
            -H "X-GitHub-Api-Version: 2026-03-10" ₩
            https://api.github.com/repos/E2IDLE/DirectP2P/dispatches ₩
            -d '{
            "event_type":"submodule_updated",
              "client_payload": {
                "branch": "${{ github.ref_name }}"
              }
            }'
```

VIII. 보안 가이드

8.1 민감 정보 취급 주의사항

- 취약점 정보

검증된 라이브러리 사용으로 현재 식별된 취약점은 없으나, 보안 무결성을 유지하기 위해 분기별 의존성 점검을 실시할 예정입니다. 오픈소스 커뮤니티의 보안 패치 노트를 지속적으로 검토하여, 보안 취약점이 보고되는 즉시 상위 버전으로의 마이그레이션 또는 긴급 업데이트를 수행하는 선제적 대응 프로세스를 운영하겠습니다

- 보안 및 무결성 정보

○ **피어 인증 및 보안 통신:** libp2p의 보안 전송 계층을 활용하여 피어 간 고유 ID(PeerID) 기반의 인증을 수행하며, TLS 1.3 프로토콜을 통해 핸드셰이크 단계부터 모든 통신을 암호화함.

○ **전송 암호화:** QUIC 프로토콜 자체의 암호화 기능과 더불어, libp2p 수준에서 지원하는 AES-256-GCM 또는 ChaCha20-Poly1305 기반의 스트림 암호화 적용.

○ **데이터 무결성:** libp2p의 멀티스트림(multistream) 환경에서 각 메시지 프레임에 대한 SHA-256 체크섬 검증을 수행하여 네트워크 전송 중 데이터 변조를 원천 차단.

○ **리플레이 공격 방지:** QUIC의 연결 식별자(Connection ID) 및 libp2p의 시퀀스 번호 관리를 통해 패킷 재전송 공격(Replay Attack)을 방지함.

8.2 인증/인가 처리 방법

○ **사용자 인증 (Authentication)**

이메일/비밀번호 기반 로그인 → 랜덤 Token 발급 (API-002)

Token 무효화 (로그아웃, API-003), 회원 탈퇴 시 계정 및 관련 데이터 삭제

○ **API 인가 (Authorization)**

모든 보호 엔드포인트: Authorization: Bearer ⟨token⟩ 헤더

인증 미들웨어: auth_middleware.go (파일 구조에 명시됨)

CORS 정책 적용 (cors.go), Rate Limiting (E5005 – 100 req/min)

○ **TURN 자격증명 (별도 인증 체계)**

HMAC-SHA256 기반 시간 제한 임시 자격증명

username = 만료 Unix Timestamp:userId

password = Base64(HMAC-SHA256(secret_key, username)), 유효 시간 1시간

○ **전송 보안**

종단 간 AES-256-GCM 암호화, TLS 1.3 (QUIC 내장)

HTTPS 강제 (HTTP → HTTPS 리다이렉트)

○ **에이전트 통신 보안**

localhost 바인딩만 허용 (외부 접근 차단)

에이전트 등록 시 API Server 토큰으로 디바이스 등록