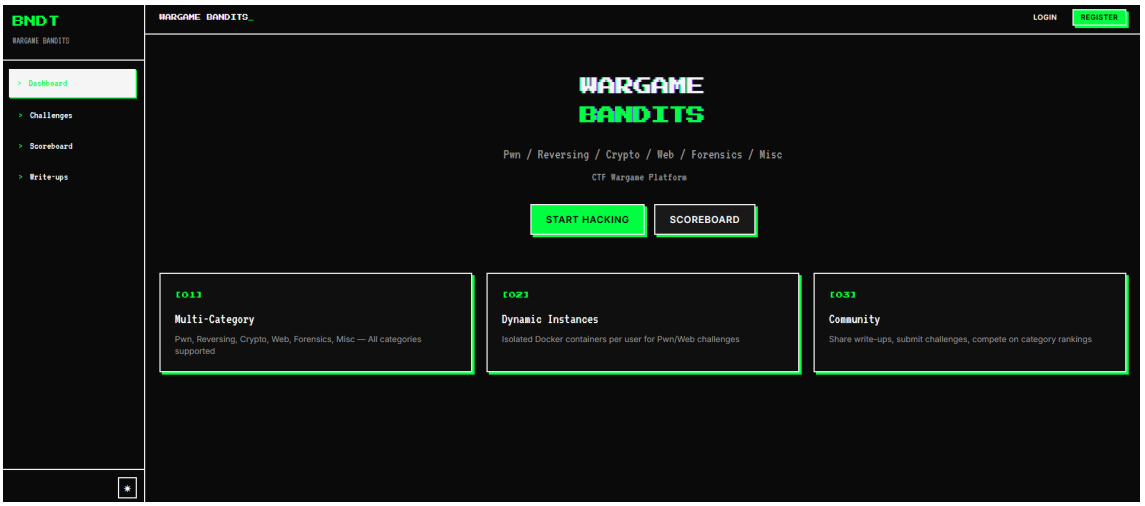
# 워게임 유괴단 프로젝트 최종 보고서

| | |
|---|---|
| 날짜 | @2026년 2월 10일 |
| 다중 선택 | 보고서 |




최종적으로 네오 브루탈리즘으로 디자인한 워게임유괴단 사이트

## 1. 프로젝트 개요

### 프로젝트명

**워게임 유괴단 (Wargame Bandits)** — 참여형 워게임 학습 플랫폼

### 프로젝트 주제

워게임 학습 및 참여형 워게임 사이트 제작

### 프로젝트 목적

- **정기적인 워게임 학습**: 드림핵을 통한 주기적인 기초 워게임 학습
- **참여형 플랫폼 구축**: IT대학 전공 과목(웹 프로그래밍, 시스템 보안, 네트워크, 암호학 등)에서 배운 지식을 활용하여 부원들이 직접 워게임 문제를 출제하고 공유
- **동아리 워게임 아카이브**: ASC만의 특색있는 워게임 아카이브를 구축하여 신입 부원들의 학습 자료로 활용

### 차별점

| 차별점 | 설명 |
|---|---|
| **학교 특화 콘텐츠** | IT대학 커리큘럼과 연계된 문제 출제 (과목 태그 시스템) |
| **부원 참여형** | 인증된 사용자가 직접 문제 제작 가능, 관리자 리뷰 후 게시 |
| **ASC CTF 아카이브** | 기존 동아리 CTF 문제들을 체계적으로 보관 및 카테고리화 |
| **단계적 학습 경로** | 난이도별(1~5단계) 체계적인 학습 경로 제공 |
| **동적 스코어링** | 풀이자 수에 따라 점수가 변동되는 동적 점수 시스템 |

### 벤치마킹 사이트

- pwnable.kr — Pwn 특화 워게임
- dreamhack.io — 종합 워게임 학습 플랫폼
- webhacking.kr — 웹 해킹 워게임

---

## 2. 기술 스택

### Backend

| 기술 | 버전 | 용도 |
|---|---|---|
| FastAPI | 0.115.6 | 비동기 웹 프레임워크 |
| Python | 3.11+ | 백엔드 언어 |
| SQLAlchemy | 2.0 (async) | ORM (비동기 DB 연동) |
| PostgreSQL | 16 | 메인 데이터베이스 |
| Redis | 7 | 캐시 / 세션 관리 |
| Celery | 5.4.0 | 비동기 태스크 큐 (컨테이너 관리) |
| Docker SDK | 7.1.0 | 동적 문제 컨테이너 관리 |
| JWT (python-jose) | - | 인증 토큰 |
| Alembic | - | DB 마이그레이션 |
| Pydantic | 2.10.5 | 데이터 검증 |

### Frontend

| 기술 | 버전 | 용도 |
|---|---|---|
| React | 18.3.1 | UI 프레임워크 |
| TypeScript | 5.7.3 | 타입 안전성 |
| Vite | 6.0.7 | 빌드 도구 |
| Zustand | 5.0.3 | 상태 관리 |
| Tailwind CSS | 3.4.17 | 스타일링 |
| React Router | 7.1.1 | 클라이언트 라우팅 |
| Axios | 1.7.9 | HTTP 클라이언트 |
| xterm.js | 5.3.0 | 웹 터미널 (Pwn 문제용) |
| Recharts | 2.15.0 | 차트/그래프 |
| react-markdown | - | 마크다운 렌더링 |

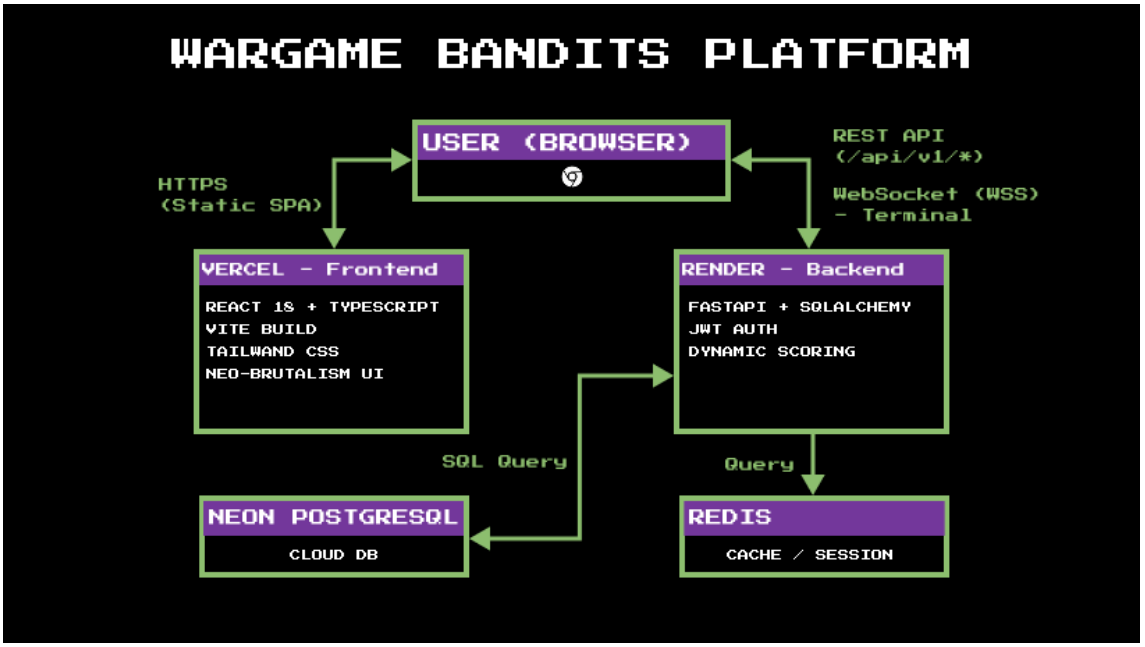
### Infrastructure

| 기술 | 용도 |
|---|---|
| Docker + Docker Compose | 컨테이너 오케스트레이션 |
| Nginx | 리버스 프록시 / 정적 파일 서빙 |
| Vercel | 프론트엔드 배포 |
| Render | 백엔드 API 배포 |

| 기술 | 용도 |
|---|---|
| Neon PostgreSQL | 클라우드 DB (Render 연동) |

## 3. 시스템 아키텍처

### 전체 구조



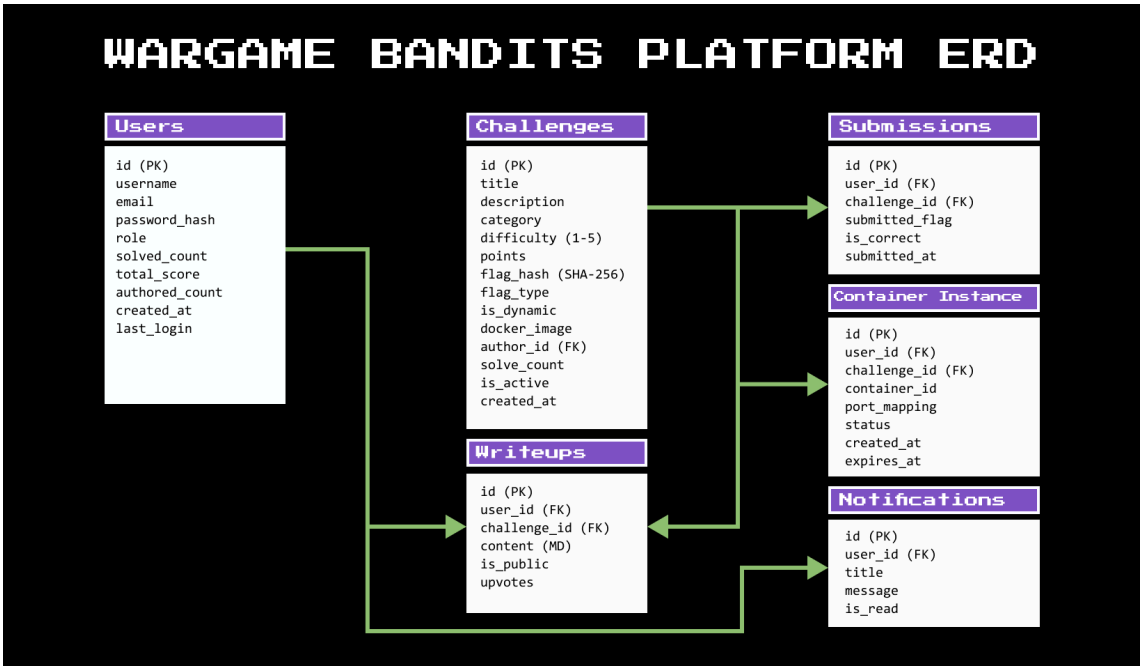

### API 엔드포인트 구조

```
/api/v1/
├── auth/
│   ├── POST /register          # 회원가입
│   ├── POST /login             # 로그인
│   └── POST /refresh           # 토큰 갱신
├── challenges/
│   ├── GET  /                  # 문제 목록 (필터/검색/페이지네이션)
│   ├── GET  /{id}              # 문제 상세
│   └── POST /{id}/submit       # 플래그 제출
├── users/
│   ├── GET  /me                # 내 프로필
│   ├── GET  /me/stats          # 내 통계
│   ├── GET  /me/heatmap        # 활동 히트맵
│   └── GET  /me/dashboard      # 대시보드 데이터
├── scoreboards/
│   ├── GET  /                  # 종합 랭킹
│   └── GET  /{category}        # 카테고리별 랭킹
├── containers/
│   ├── POST /                  # 인스턴스 생성
│   ├── GET  /{id}              # 인스턴스 상태
│   └── DELETE /{id}            # 인스턴스 종료
```

```
├── writeups/
│   ├── GET  /                  # Write-up 목록
│   └── POST /                  # Write-up 작성
├── admin/
│   ├── POST /challenges        # 문제 생성
│   ├── PUT  /challenges/{id}   # 문제 수정
│   ├── DELETE /challenges/{id} # 문제 삭제
│   └── POST /challenges/import # YAML 일괄 등록
└── ws/terminal/{instance_id}   # WebSocket 터미널
```

# 4. 데이터베이스 설계

## ERD (Entity Relationship)

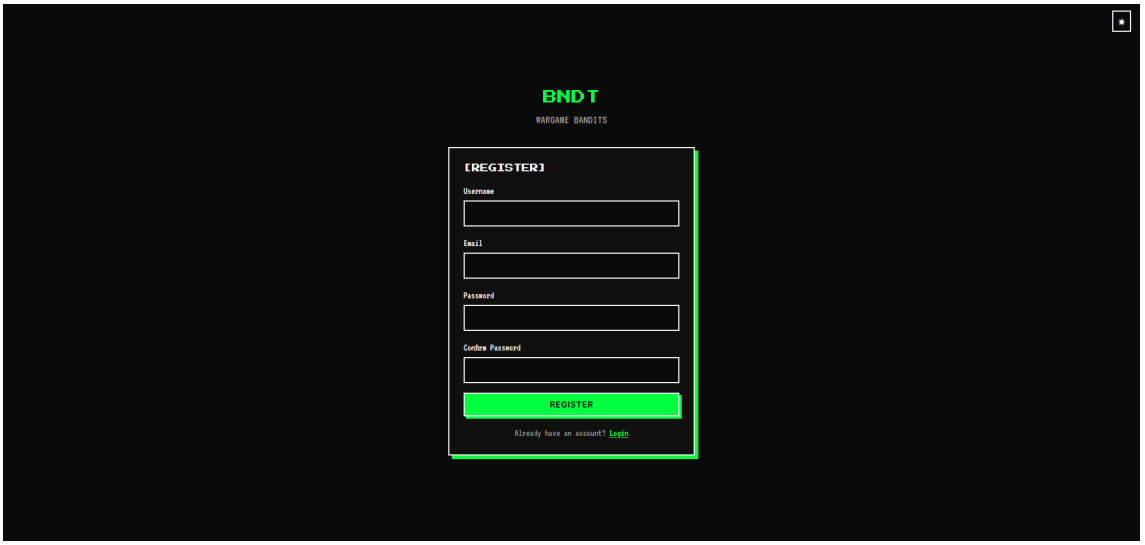



## 주요 테이블 (6개)

| 테이블 | 역할 | 주요 필드 |
|---|---|---|
| **Users** | 사용자 정보 | username, email, role(user/admin/author), total_score |
| **Challenges** | 워게임 문제 | title, category, difficulty, flag_hash, is_dynamic |
| **Submissions** | 플래그 제출 기록 | user_id, challenge_id, is_correct, submitted_at |
| **ContainerInstances** | Docker 인스턴스 | container_id, port_mapping, status, expires_at |
| **Writeups** | 풀이 Write-up | content(Markdown), is_public, upvotes |
| **Notifications** | 알림 | title, message, is_read |

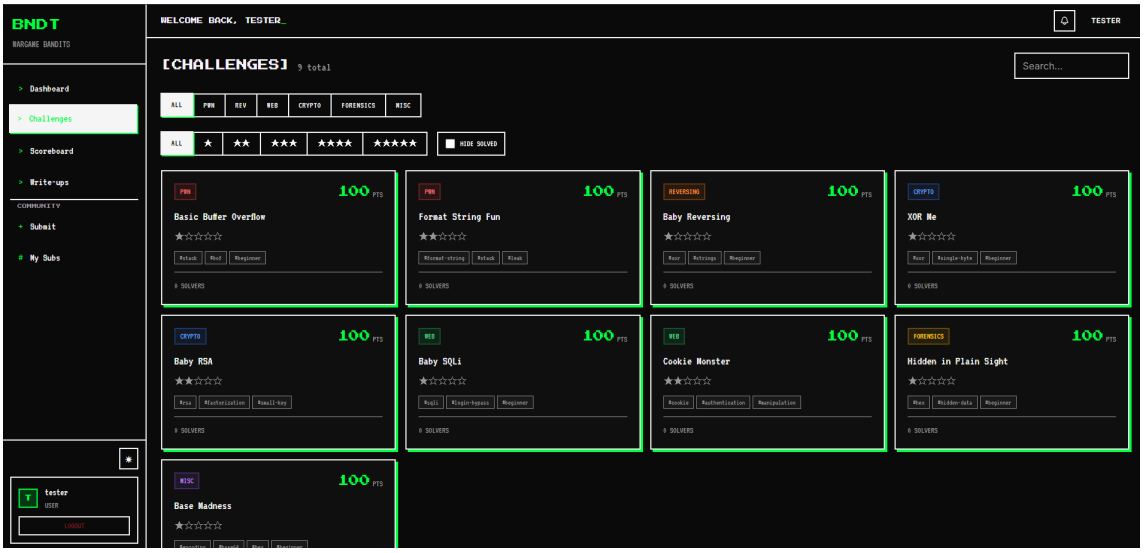
# 5. 주요 기능 구현

## 5.1 인증 시스템




- **bcrypt** 기반 비밀번호 해싱 (평문 저장 절대 금지)
- **JWT** 토큰 방식: Access Token (15분) + Refresh Token (7일)
- **역할 기반 접근 제어(RBAC)**: user, admin, challenge_author 3단계

## 5.2 챌린지 시스템

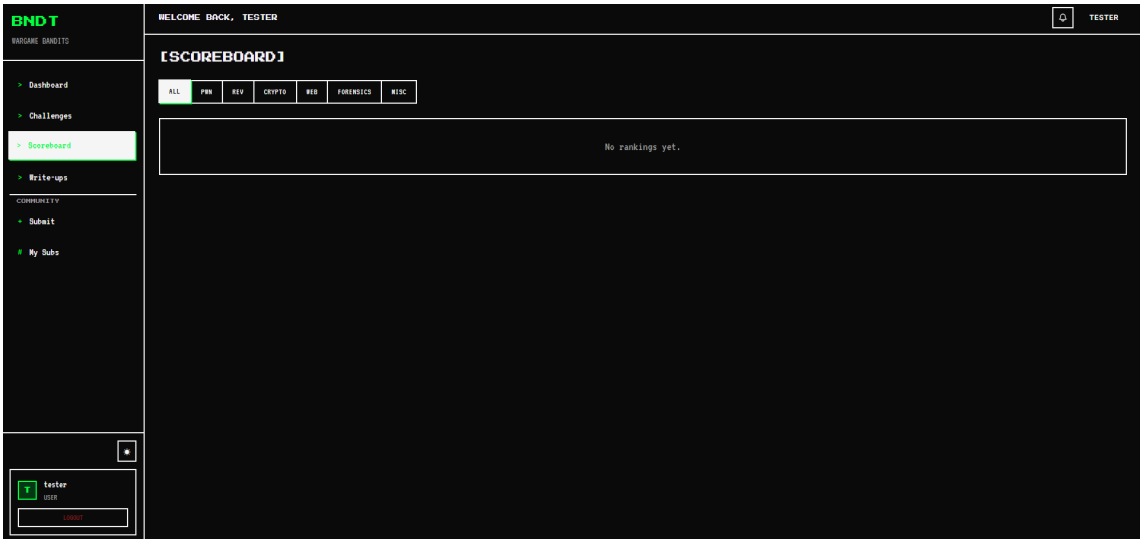



- **6개 카테고리** 지원: Pwn, Reversing, Crypto, Web, Forensics, Misc
- **5단계 난이도**: Baby → Easy → Medium → Hard → Insane
- **동적 스코어링**: `points = max(min_points, max_points - decay × solve_count)`
- **플래그 검증**: SHA-256 해시 비교 (평문 플래그 DB 저장 금지)
- **중복 제출 방지**: 이미 풀이한 문제 재제출 차단
- **Rate Limiting**: 플래그 제출 분당 10회 제한
- **YAML 일괄 등록**: challenge.yaml 파일로 문제 대량 등록

## 5.3 동적 인스턴스 (Docker 컨테이너)

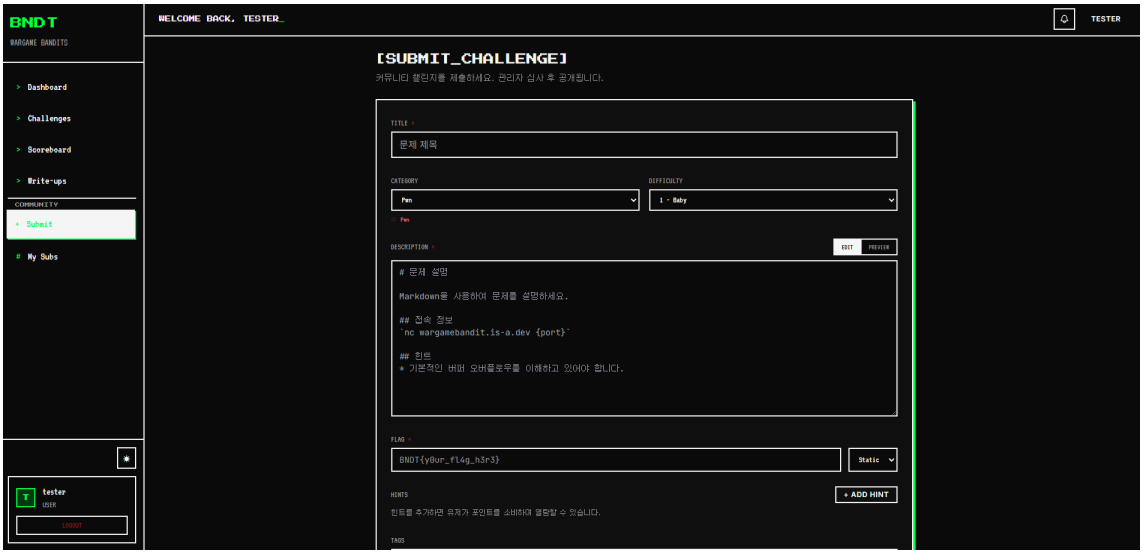
- 유저별 격리된 Docker 컨테이너 생성 (Pwn/Web 문제)
- 포트 범위: 30000~39999 동적 할당
- 리소스 제한: CPU 0.5코어, RAM 128MB
- 자동 만료: 30분 타임아웃, Celery Beat으로 5분마다 정리
- 유저당 동시 실행 최대 3개
- xterm.js 기반 웹 터미널 연동 (WebSocket)

## 5.4 스코어보드




- 종합 랭킹 + 카테고리별 개별 랭킹
- 동적 점수제 반영 실시간 순위 갱신

## 5.5 커뮤니티 기능




- **Write-up 시스템**: 풀이한 문제만 Write-up 작성 가능 (스포일러 방지)

- **문제 제작 참여**: 인증된 사용자가 문제 제출 → 관리자 리뷰 후 게시
- **알림 시스템**: 신규 문제, First Blood, 리뷰 결과 등 실시간 알림

### 5.6 대시보드


- GitHub 스타일 활동 히트맵 (연간 풀이 현황)
- 종합 통계 카드 (랭킹, 점수, 풀이 수, 스트릭)
- 카테고리별 레이더 차트
- 추천 문제 목록
- 최근 활동 내역

---

## 6. UI/UX 디자인

### 디자인 시스템: Neo-Brutalism

CTF/해커 문화에 어울리는 **네오 브루탈리즘(Neo-Brutalism)** 디자인 시스템을 자체 구축했다.

### 디자인 특징

| 요소 | 설명 |
|---|---|
| **보더** | 2px solid, 라운딩 없음 (0px) — 날카롭고 각진 느낌 |
| **그림자** | 하드 오프셋 (4px 4px 0) — 입체감 있는 브루탈리즘 특유의 그림자 |
| **호버 효과** | translate -2px + 그림자 확대 — 요소가 떠오르는 느낌 |
| **액티브 효과** | translate +2px + 그림자 제거 — 눌리는 피드백 |
| **애니메이션** | 깜빡이는 커서, 타이프라이터 텍스트 효과 |

### 컬러 팔레트

| 용도 | 라이트 모드 | 다크 모드 |
|---|---|---|
| **배경** | #FFFFFF (순백) | #0A0A0A (거의 순흑) |
| **텍스트** | #000000 | #FFFFFF |

| 용도 | 라이트 모드 | 다크 모드 |
|---|---|---|
| **네온 악센트** | #00FF41 (네온 그린) | #00FF41 (네온 그린) |
| **프라이머리** | #7C3AED (퍼플) | #7C3AED (퍼플) |
| **보더/그림자** | #000000 | #FFFFFF / #00FF41 |

### 타이포그래피

| 용도 | 폰트 | 설명 |
|---|---|---|
| **제목** | Press Start 2P | 픽셀 폰트 — 레트로 게임 느낌 |
| **데이터/라벨** | VT323 | 모노스페이스 레트로 — 터미널 느낌 |
| **본문** | Inter | 깔끔한 가독성 — 긴 텍스트용 |

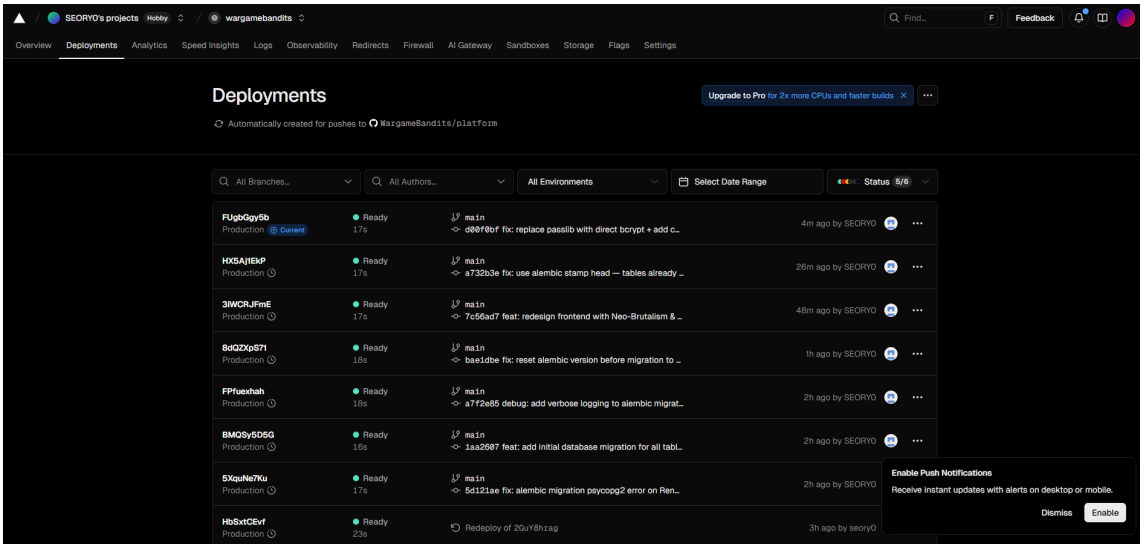
### 커스텀 UI 컴포넌트 (6종)

- **BrutalButton**: primary / neon / secondary 3가지 변형
- **BrutalCard**: 하드 그림자 + 두꺼운 보더 카드
- **BrutalInput**: 브루탈리즘 스타일 입력 필드
- **BrutalBadge**: 카테고리/난이도 태그
- **BrutalTable**: 지브라 스트라이프 데이터 테이블
- **BrutalTabs**: 탭 내비게이션

---

## 7. 배포 과정 — 삽질의 기록

### 최초 계획: GitHub Pages + Vercel



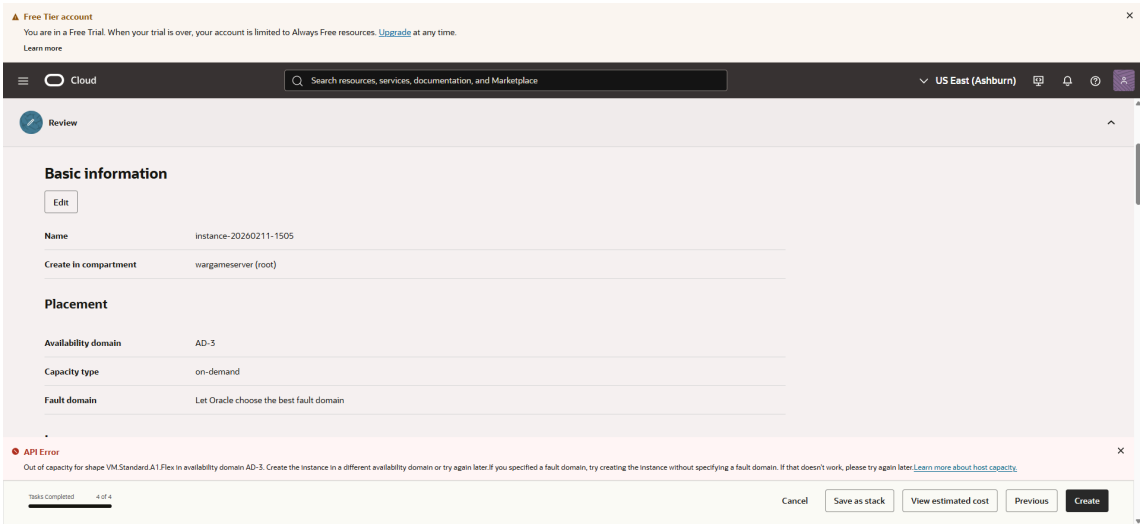

Vercel Delpoyments

처음에는 비용을 최소화하기 위해 **GitHub Pages**로 프론트엔드를, **Vercel Serverless Functions**로 백엔드를 배포하려 했다.

**실패 원인:**

- **GitHub Pages**는 정적 사이트 호스팅 전용으로, React SPA의 클라이언트 사이드 라우팅을 제대로 지원하지 못했다. 새로고침 시 404 에러가 발생했고, 우회 방법(404.html 리다이렉트)은 SEO 문제와 깜빡임을 유발했다.
- **Vercel Serverless Functions**로 FastAPI를 래핑하려 했으나, FastAPI의 비동기 생태계(SQLAlchemy async, Celery, Docker SDK)가 Serverless 환경과 호환되지 않았다. Cold start 문제, WebSocket 미지원, PostgreSQL 커넥션 풀 관리 불가 등 근본적인 한계에 부딪혔다.
- 특히 Docker 컨테이너 기반 동적 인스턴스 기능은 Serverless 환경에서 구현 자체가 불가능했다.

## 두 번째 시도: Oracle Cloud Free Tier (자체 호스팅)




oracle cloud api error: out of capacity 오류

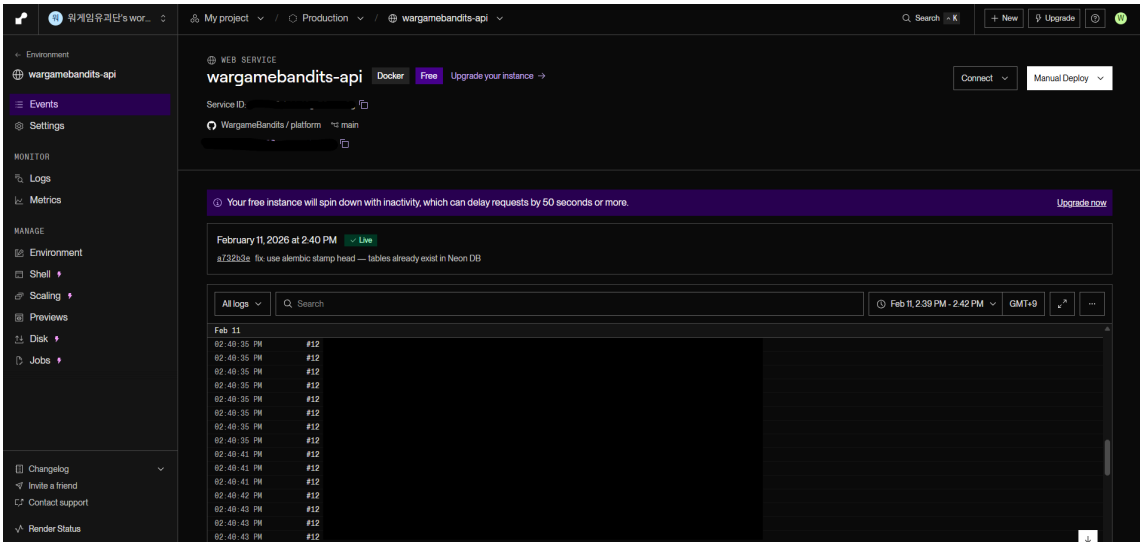
이후 차선책으로 **Oracle Cloud Free Tier**(ARM Ampere A1, 4 OCPU, 24GB RAM)에 전체 스택을 자체 호스팅하려 했다.

docker-compose.prod.yml, Nginx SSL 설정, Let's Encrypt 인증서 발급 스크립트(init-letsencrypt.sh), 서버 초기 세팅 스크립트(setup-server.sh)까지 모두 준비했다.

**실패 원인:**

- 한국 리전에서는 리소스 가용성 부족으로 인스턴스 생성이 거의 불가능한 상황이었다.
- 다른 지역으로도 여러 번 호스팅을 시도했으나 "Out of Capacity" 에러가 반복되었다.
- 무료 서버를 확보하지 못해 자체 호스팅 전략을 포기해야 했다.

## 최종 선택: Vercel + Render 조합




Render 설정 창

여러 무료 클라우드 서비스를 검토한 끝에 **Vercel(프론트엔드) + Render(백엔드)** 조합으로 최종 결정했다.

**선택 이유:**

| 서비스 | 역할 | 무료 플랜 장점 |
|---|---|---|
| **Vercel** | 프론트엔드 (React SPA) | GitHub 연동 자동 배포, 글로벌 CDN, SPA 라우팅 완벽 지원 |
| **Render** | 백엔드 (FastAPI API) | Docker 지원, 무료 PostgreSQL(Neon), 자동 HTTPS |

**아키텍처 변경 사항:**

- 프론트엔드와 백엔드가 다른 도메인에 배포되므로 **CORS 설정**을 조정했다.
- Docker 기반 동적 인스턴스 기능은 Render 무료 플랜에서 Docker-in-Docker를 지원하지 않아 **비활성화**(DOCKER_ENABLED=false)했다. 이 기능은 추후 자체 서버 확보 시 활성화할 수 있도록 **Feature Flag** 방식으로 구현해두었다.
- Celery 태스크 큐 역시 무료 플랜 제약으로 **비활성화**(CELERY_ENABLED=false)했다.
- 정적 문제(Crypto, Reversing, Forensics, Misc 등 플래그를 직접 입력하는 유형)는 정상 동작하며, 동적 문제(Pwn, Web 컨테이너)는 자체 서버 확보 후 활성화 예정이다.

**배포 구성도:**

```
사용자 ──→ Vercel (*.vercel.app)
           │ React SPA
           │ 정적 파일 CDN
           │
           └──→ API 호출 ──→ Render (*.onrender.com)
                             │ FastAPI
                             │ JWT 인증
                             │ 챌린지 CRUD
                             │
                             └──→ Neon PostgreSQL (클라우드 DB)
```

## 배포 과정에서 배운 점

1. **Serverless ≠ 만능**: 상태를 유지해야 하는 애플리케이션(WebSocket, Docker 컨테이너, DB 커넥션 풀)은 Serverless 환경과 근본적으로 맞지 않는다.
2. **무료 인프라의 한계**: Oracle Cloud Free Tier의 가용성 문제처럼, 무료 서비스는 안정성이 보장되지 않는다. 백업 플랜을 항상 준비해야 한다.
3. **Feature Flag의 가치**: Docker, Celery 같은 인프라 의존 기능을 Feature Flag로 분리해둔 덕분에, 배포 환경에 따라 유연하게 기능을 켜고 끌 수 있었다.
4. **프론트/백 분리 배포의 장점**: 각각 최적화된 플랫폼에 배포하면 비용과 성능 모두에서 이점이 있다.

---

## 8. 프로젝트 구조

```
wargame-bandits/
├── design.md                          # Neo-Brutalism 디자인 시스템 명세
├── docker-compose.yml                 # 개발 환경
├── docker-compose.prod.yml            # 프로덕션 환경 (자체 호스팅용)
├── render.yaml                        # Render 배포 설정
├── .env.example                       # 환경변수 템플릿
├── setup-server.sh                    # 서버 초기 세팅 스크립트
├── init-letsencrypt.sh                # SSL 인증서 발급 스크립트
│
├── backend/                           # FastAPI 백엔드 (50개 Python 파일)
│   ├── app/
│   │   ├── main.py                    # FastAPI 엔트리포인트
│   │   ├── config.py                  # Pydantic Settings 환경 설정
│   │   ├── database.py                # 비동기 SQLAlchemy 연결
│   │   ├── celery_app.py              # Celery 태스크 큐
│   │   ├── api/v1/                    # API 라우터 (8개)
│   │   ├── models/                    # SQLAlchemy 모델 (6개)
│   │   ├── schemas/                   # Pydantic 스키마 (7개)
│   │   ├── services/                  # 비즈니스 로직 (8개)
│   │   ├── core/                      # 보안, 권한, 예외 처리
│   │   └── tasks/                     # Celery 비동기 태스크
│   ├── challenges/                    # 샘플 CTF 문제 (9개)
│   ├── alembic/                       # DB 마이그레이션
│   ├── requirements.txt               # Python 의존성 (36개)
│   ├── Dockerfile                     # 개발용 Docker 이미지
│   └── Dockerfile.prod                # 프로덕션용 (Gunicorn)
│
├── frontend/                          # React 프론트엔드 (61개 TypeScript 파
일)
│   ├── src/
│   │   ├── pages/                     # 14개 페이지
│   │   ├── components/                # 25+ 컴포넌트
│   │   │   ├── ui/                    # 커스텀 Brutal UI 컴포넌트 (6종)
│   │   │   ├── layout/                # 레이아웃 (Sidebar, TopBar)
│   │   │   ├── challenge/             # 챌린지 관련
│   │   │   ├── dashboard/             # 대시보드 (히트맵, 통계)
```

```
│   │   │   └── terminal/            # 웹 터미널 (xterm.js)
│   │   ├── stores/                  # Zustand 상태 관리
│   │   ├── services/                # API 클라이언트 (11개)
│   │   ├── types/                   # TypeScript 인터페이스
│   │   └── utils/                   # 유틸리티 함수
│   ├── package.json                 # 프론트엔드 의존성 (30+)
│   ├── vite.config.ts
│   ├── tailwind.config.ts           # Neo-Brutalism 테마
│   └── Dockerfile
│
├── nginx/                           # Nginx 설정
│   ├── nginx.dev.conf               # 개발 환경 (HTTP)
│   └── nginx.prod.conf              # 프로덕션 (HTTPS + SSL)
│
└── certbot/                         # Let's Encrypt 인증서 (프로덕션)
```

## 9. 보안 설계

| 보안 항목 | 구현 방식 |
|---|---|
| **비밀번호 저장** | bcrypt 해싱 (평문 저장 금지) |
| **인증 토큰** | JWT Access Token (15분) + Refresh Token (7일) |
| **플래그 저장** | SHA-256 해시 (평문 플래그 DB 저장 금지) |
| **Rate Limiting** | 플래그 제출 분당 10회 제한 |
| **Anti-Bruteforce** | 연속 틀린 제출 시 쿨다운 적용 |
| **입력 검증** | 모든 입력값 Pydantic 스키마 검증 |
| **SQL Injection** | ORM 사용 필수, raw query 금지 |
| **XSS 방지** | 마크다운 렌더링 시 sanitize 처리 |
| **CORS** | 허용 Origin 명시적 지정 |
| **HTTPS** | 프로덕션 환경 SSL 강제 |
| **보안 헤더** | X-Frame-Options, HSTS, CSP 설정 |
| **컨테이너 격리** | 네트워크 격리, capability 최소화, read-only FS |

## 10. 샘플 문제 목록

프로젝트에 포함된 9개의 샘플 워게임 문제:

| # | 카테고리 | 문제명 | 난이도 | 배점 | 설명 |
|---|---|---|---|---|---|
| 1 | Pwn | Basic Buffer Overflow | 1 | 100 | gets() 취약점을 이용한 스택 BOF |
| 2 | Pwn | Format String Attack | 2 | 200 | 포맷 스트링 취약점 공격 |
| 3 | Web | SQL Injection 101 | 1 | 150 | 기본 SQL 인젝션 |
| 4 | Web | Cookie Monster | 1 | 100 | 쿠키 조작 문제 |
| 5 | Reversing | Baby Rev | 1 | 100 | 기초 리버싱 |

| # | 카테고리 | 문제명 | 난이도 | 배점 | 설명 |
|---|---|---|---|---|---|
| 6 | Crypto | Baby RSA | 1 | 100 | RSA 기초 문제 |
| 7 | Crypto | XOR Cipher | 1 | 100 | XOR 암호화 |
| 8 | Forensics | Hidden Message | 1 | 100 | 스테가노그래피 |
| 9 | Misc | Base Madness | 1 | 100 | 인코딩 문제 |

## 11. 프로젝트 통계

| 항목 | 수치 |
|---|---|
| 총 파일 수 | ~180개 |
| 총 코드 라인 수 | ~10,000줄 이상 |
| Backend Python 파일 | 50개 |
| Frontend TypeScript 파일 | 61개 |
| API 엔드포인트 | ~30개 |
| DB 테이블 | 6개 |
| UI 페이지 | 14개 |
| 커스텀 컴포넌트 | 25개 이상 |
| 샘플 문제 | 9개 (6 카테고리) |
| Backend 의존성 | 36개 |
| Frontend 의존성 | 30개 이상 |

## 12. 향후 계획

| 우선순위 | 항목 | 설명 |
|---|---|---|
| 높음 | 자체 서버 확보 | Docker 동적 인스턴스 활성화를 위한 서버 확보 |
| 높음 | 문제 확충 | IT대학 커리큘럼 연계 문제 30개 이상 제작 |
| 중간 | ASC CTF 아카이브 | 기존 동아리 CTF 문제 이관 및 정리 |
| 중간 | 부원 참여 활성화 | 문제 제작 가이드 작성, 기여자 인센티브 시스템 |
| 낮음 | CI/CD 파이프라인 | GitHub Actions 자동 배포 구축 |
| 낮음 | 모바일 최적화 | 반응형 디자인 고도화 |

## 13. 결론

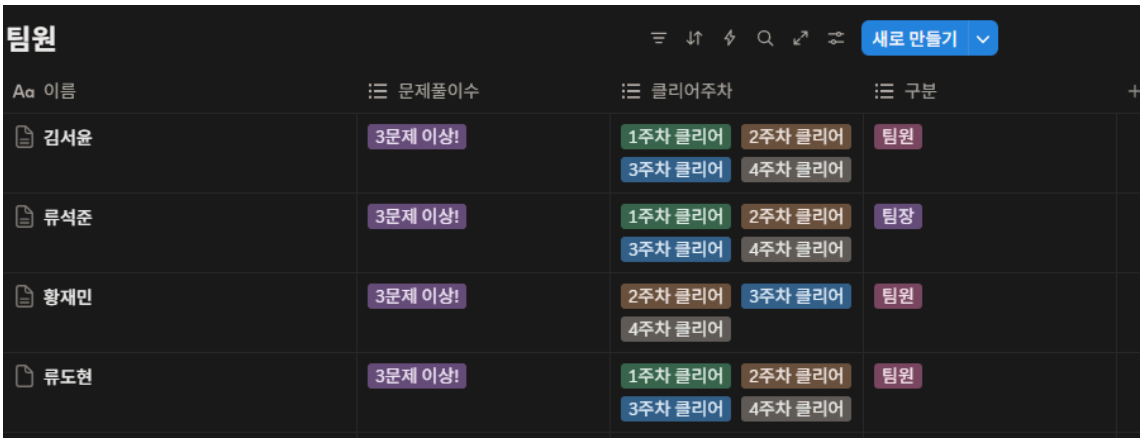
워게임 유괴단 프로젝트는 단순한 CTF 문제 풀이 사이트를 넘어, **부원들이 직접 문제를 출제하고 공유하는 참여형 워게임 플랫폼**을 목표로 했다. 프론트엔드(React + TypeScript)와 백엔드(FastAPI + PostgreSQL)를 완전히 분리한 아키텍처를 설계하고, Neo-Brutalism이라는 독특한 디자인 시스템을 자체 구축하여 CTF 문화에 어울리는 시각적 정체성을 확립했다.

배포 과정에서 GitHub Pages, Oracle Cloud Free Tier 등 여러 대안을 시도하며 실패를 겪었지만, 최종적으로 Vercel + Render 조합을 통해 **완전 무료**로 플랫폼을 운영할 수 있는 인프라를 확보했다. 이 과정에서 Feature Flag 패턴을 도입하여 배포 환경에 따라 기능을 유연하게 제어할 수 있도록 설계한 것이 핵심 성과 중 하나다.

6개 CTF 카테고리, 동적 스코어링, Docker 기반 동적 인스턴스, WebSocket 터미널, Write-up 커뮤니티, 활동 히트맵 대시보드 등 워게임 플랫폼에 필요한 핵심 기능을 모두 구현하였으며, 향후 자체 서버 확보 시 Docker 동적

인스턴스 기능을 활성화하여 Pwn/Web 문제의 실시간 환경까지 제공할 수 있는 기반을 마련해두었다.

## Ex. 워게임 학습




프로젝트 기간 동안 팀원 4명 전원이 dreamhack.io를 통해 **4주간 주 3문제 이상** 워게임 풀이를 진행했다. 플랫폼 개발과 병행하여 실제 워게임 문제를 풀어봄으로써 사용자 관점의 요구사항을 파악하고, 자체 플랫폼에 출제할 문제의 난이도와 형식을 설계하는 데 기반 자료로 활용했다. 팀원 4명 중 3명이 CTF 초보자임을 감안할 때 학습 성과가 뛰어났다고 할 수 있다.