DXCMS BOARD v8.0.3

미니 프레임워크 · 엔진 구조 상세 설명서

3.1 엔진 개요 · DX 엔진 구조

부트스트랩 · 라우터 · DI 컨테이너 · 훅 시스템 · 보안 · 캐시 · 테마 엔진

DesignOneX · 2025

1. DXCMS 미니 프레임워크 개요

DXCMS는 단일 진입점(index.php)을 중심으로 한 라라벨(Laravel) 철학 기반의 경량 PHP 프레임워크를 내장하고 있습니다. 외부 프레임워크에 의존하지 않고, PHP 5.6부터 8.x까지 단일 코드베이스로 동작하는 자체 엔진을 구현했습니다.

설계 철학

- 단일 진입점 (Front Controller 패턴) — 모든 요청을 index.php 하나가 받는다
- 관심사 분리 — Secure(보안), Router(라우팅), Auth(인증), Cache(캐시) 완전 분리
- 폴백 설계 — Redis → APCu → 파일, 테마 스킨 → 기본 스킨 등 항상 동작을 보장
- PHP 5.6 완전 호환 — 클로저, ?? 연산자, 반환 타입 힌트 없이 구현
- 라라벨 스타일 API — DxRouter, DxContainer, QueryBuilder 등 익숙한 인터페이스 제공

1.1 핵심 상수

상수명	값 / 설명
DX_CMS	true — 직접 접근 차단용 가드
DX_VERSION	'8.0.3' — 현재 CMS 버전
DX_ROOT	index.php 위치 절대 경로
DX_CORE	DX_ROOT/core/ — 엔진 클래스 루트
DX_DATA	DX_ROOT/data/ — 런타임 데이터(캐시·업로드)
DX_THEMES	DX_ROOT/themes/ — 테마 루트
DX_PLUGINS	DX_ROOT/plugins/ — 플러그인 루트
DX_BOARDS	DX_ROOT/boards/ — 게시판 핸들러
DX_START	microtime(true) — 요청 시작 시각 (성능 측정용)
DX_SECRET_KEY	설치 시 생성된 64자리 랜덤 키 — 세션/CSRF 키 도출
DX_SECURITY_PATH	core/security/{16자리해시}/ — Secure.php 난독화 경로

2. 부트스트랩 — 요청 처리 전체 흐름

index.php는 모든 HTTP 요청의 단일 진입점입니다. 다음 6단계로 순차 실행되며, 각 단계가 완전히 완료된 후 다음 단계로 진행됩니다.

```
// ═══════════════════════════════════════════════
// 요청 처리 전체 흐름 (index.php 실행 순서)
// ═══════════════════════════════════════════════


[STEP 0]  ob_start() + URL 정규화 + PHP 버전 체크
[STEP 1]  클래스/함수 파일 로드 (require_once)
[STEP 2]  보안 초기화 (Secure — 세션·CSRF·헤더)
[STEP 3]  DB 연결 + 설정 로드 (config.php)
[STEP 3B] 시크릿 키 주입 + 세션/CSRF 키 도출
[STEP 4]  핵심 서비스 초기화 (훅·플러그인·사이트·테마·인증·DI)
[STEP 5]  라우팅 + 디스패치 → 핸들러 실행
[STEP 6]  테마 레이아웃 렌더링 → 응답 출력
```

2.1 STEP 0 — 사전 처리

ob_start()와 URL 정규화

- ob_start(): IIS/CGI/웹호스팅에서 'headers already sent' 오류 없이 header() 동작 보장
- 이중 슬래시 정규화: //path → /path 로 301 리다이렉트 (SEO 중복 URL 방지)
- PHP 버전 체크: 5.6 미만이면 즉시 오류 메시지 출력 후 종료
- 미설치 감지: data/config.php 없으면 install/ 로 자동 리다이렉트

2.2 STEP 1 — 클래스 로드 순서

실행 없이 클래스·함수 정의만 메모리에 올립니다. 의존 관계에 따라 순서가 엄격히 정해져 있습니다.

로드 순서	파일	역할
1	core/functions.php	전역 헬퍼 함수 (dx_esc, dx_csrf_*, dx_config 등)

로드 순서	파일	역할
2	core/security/{hash}/Secure.php	보안 클래스 — 난독화 경로 우선, 없으면 core/Secure.php
3	core/DxSanitizer.php	입력 정제·HTML 필터
4	core/db/Database.php	PDO 래퍼 (연결 전)
5	core/hook/HookManager.php	훅 시스템
6	core/PluginRegistry.php	플러그인 레지스트리
7	core/auth/Auth.php	세션 인증
8	core/DxSite.php	멀티사이트 관리
9	core/DxTheme.php	테마 엔진
10	core/DxCache.php	멀티 드라이버 캐시
11	core/DxSeo.php	SEO 헬퍼
12	core/DxRouter.php	라라벨 스타일 라우터
13	core/DxContainer.php	DI 컨테이너
14	core/db/QueryBuilder.php	쿼리 빌더
...	기타 코어 클래스	DxCategory, DxPoint, DxNotification 등

2.3 STEP 2 — 보안 초기화

Secure::getInstance()가 세션 설정, 세션 시작, 보안 헤더 발행, CSRF 토큰 발급을 순서대로 처리합니다. 특히 비로그인 GET 요청에서는 세션을 시작하지 않아 파일 락을 제거하고 동시 처리 성능을 향상시킵니다.

```
// 세션 최적화: 비로그인 GET 요청은 세션 시작 안 함
if (GET 요청 AND 세션 쿠키 없음 AND AJAX 아님) {
    // /admin, /auth, /view/, /api/, /write 제외
    $_dxNeedSession = false;  // 파일락 제거 → 동시처리 성능 향상
}
if ($_dxNeedSession) Secure::startSession();
Secure::sendSecurityHeaders();  // X-Frame-Options 등
Secure::csrfToken();            // CSRF 토큰 선제 발급
```

2.4 STEP 3 — DB 연결 + 시크릿 키

data/config.php를 include하면 $db->connect()가 실행되어 PDO 연결이 완성됩니다. 이후 DX_SECRET_KEY를 기반으로 세션 키 이름과 CSRF 키 이름을 동적으로 도출합니다. 소스코드가 공개되어도 키 이름을 예측할 수 없습니다.

```
// 시크릿 키 기반 동적 키 도출
// config.php에 저장된 64자리 랜덤 값 사용
$secretKey = dx_config('secret_key', '');
Secure::initSecretKeys($secretKey);
// 결과: $keySession = substr(sha1('dx_user'.$secretKey), 0, 12);
// 결과: $keyCsrf    = substr(sha1('dx_csrf'.$secretKey), 0, 12);
// → 사이트마다 고유한 세션/CSRF 키 이름 → 예측 불가
```

2.5 STEP 4 — 핵심 서비스 초기화

DB 연결 완료 후 모든 핵심 서비스가 초기화됩니다. 초기화 순서가 의존 관계를 따릅니다.

초기화 대상	의존	설명
HookManager::getInstance()	없음	훅 시스템 준비 (플러그인이 훅 등록 전에 준비되어야 함)
load_plugins()	HookManager	plugins/ 폴더 스캔 → 활성 플러그인 plugin.php 실행
DxSite::getInstance()	Database	도메인 감지 → dx_config 전역 오버라이드
DxTheme::getInstance()	DxSite	테마명 확정 (DxSite 이후에야 도메인별 테마 알 수 있음)
Auth::getInstance()	Database, Secure	세션에서 사용자 검증 (DB 연결 완료 후 가능)
DxContainer::registerCoreServices()	전체	DI 컨테이너에 db, auth, cache, hook 등 등록
DxExtend::runTop()	전체	extend/top/ 파일 자동 실행

3. DxRouter — 라라벨 스타일 라우터

DxRouter는 Laravel의 라우터를 PHP 5.6 호환으로 재구현한 클래스입니다. 기존 파일 기반 디스패처(Dispatcher)와 공존하며, DxRouter에 등록된 라우트가 없으면 기존 방식으로 폴백합니다.

3.1 라우트 등록 방법

```
// routes/web.php 에서 사용

// 기본 등록
DxRouter::get('/mypage/dashboard', 'MemberController@dashboard')
        ->middleware('auth');

DxRouter::post('/api/update', 'MemberController@update')
        ->middleware(array('auth', 'csrf'));

// 클로저 사용
DxRouter::get('/hello', function($params) {
    echo 'Hello, World!';
});

// 그룹 (prefix + 미들웨어 일괄 적용)
DxRouter::group(array('prefix'=>'/shop','middleware'=>'auth'), function() {
    DxRouter::get('/cart', 'ShopController@cart');
    DxRouter::post('/order', 'ShopController@order');
});

// REST 리소스 자동 등록 (index/show/store/update/destroy)
DxRouter::resource('/posts', 'PostController');
```

3.2 URI 패턴 매칭

중괄호 파라미터({id}, {slug} 등)를 정규식으로 변환하여 URL에서 값을 추출합니다.

```
// {param} → (?P<param>[^/]+) 로 변환
// 예: '/posts/{id}' + 요청 '/posts/123'  → $params['id'] = '123'
// 예: '/u/{name}/posts/{id}' + '/u/alice/posts/7' → name=alice, id=7

// 매칭 내부 로직 (matchUri 메서드)
$regex = preg_replace('/\{([a-zA-Z_][a-zA-Z0-9_]*)\}/', '(?P<$1>[^/]+)',
$pattern);
preg_match('#^'.$regex.'$#', $uri, $matches);
```

3.3 내장 미들웨어

미들웨어 이름	동작	실패 시
`auth`	로그인 여부 확인	로그인 페이지로 리다이렉트
`admin`	로그인 + 관리자 여부 확인	403 Forbidden 출력 후 종료
`guest`	비로그인 여부 확인 (회원가입 등)	홈으로 리다이렉트
`csrf`	POST CSRF 토큰 검증	403 + JSON 에러 응답
`json`	응답 헤더를 application/json으로 설정	-
`throttle`	Rate Limiting (분당 60회 기본)	429 Too Many Requests

3.4 파일 기반 폴백 디스패처

DxRouter에 일치하는 라우트가 없으면 기존 파일 기반 Dispatcher가 실행됩니다. Dispatcher는 URL 세그먼트를 분석하여 boards/handler.php, pages/, admin/, api/ 등 적절한 핸들러 파일로 라우팅합니다.

```
// STEP 5 라우팅 흐름


// 1순위: DxRouter (routes/web.php에 등록된 라우트)
if (DxRouter::dispatch()) {
    // 매칭됨 → 컨트롤러 실행 후 종료
}
// 2순위: 파일 기반 Dispatcher (기존 방식)
Dispatcher::run($uri);  // URL → 파일 매핑
// /free/view/123  → boards/handler.php (board_key=free, action=view, id=123)
// /about          → pages/ (slug=about)
// /admin          → admin/index.php
```

4. DxContainer — 의존성 주입(DI) 컨테이너

DxContainer는 Laravel의 Service Container 철학을 PHP 5.6에서 구현한 경량 IoC 컨테이너입니다. 기존 싱글턴 패턴(getInstance())과 100% 호환되면서, 플러그인에서 서비스를 등록하고 꺼내 쓸 수 있는 통합 서비스 레지스트리 역할을 합니다.

4.1 바인딩 종류

메서드	동작	사용 예
`bind()`	make() 호출마다 새 인스턴스 생성	매번 새 객체가 필요한 요청 처리기
`singleton()`	첫 make() 이후 동일 인스턴스 재사용	DB 연결, 캐시 드라이버 등
`instance()`	이미 생성된 객체를 직접 등록	CMS 초기화 완료 후 core 서비스 등록
`alias()`	별칭 등록 (db → database)	짧은 이름으로 접근

4.2 사용 방법

```
// ── 바인딩 (plugins/my-plugin/plugin.php 등에서) ──
dx_app()->singleton('mailer', function($c) {
    return new MyMailer(dx_config('smtp_host'));
});

// ── 꺼내 쓰기 ──
$mailer = dx_app()->make('mailer');  // singleton → 동일 인스턴스
$mailer = dx_make('mailer');          // 단축 함수

// ── 컨트롤러 자동 의존성 주입 ──
dx_app()->call('BoardController@index', array('slug' => 'free'));
// → BoardController 파일 자동 탐색 → 인스턴스화 → index() 호출

// ── 자동 컨트롤러 탐색 경로 ──
// controllers/{ClassName}.php
// controllers/{classname}.php  (소문자)
// core/controllers/{ClassName}.php
// plugins/*/controllers/{ClassName}.php
```

4.3 기본 등록 서비스

registerCoreServices() 메서드가 CMS 초기화 완료 후 다음 서비스를 자동 등록합니다.

서비스 이름	실제 클래스	접근 방법
`db` / `database`	Database::getInstance()	dx_make('db') 또는 dx_app()->make('database')
`auth`	Auth::getInstance()	dx_make('auth')
`secure`	Secure::getInstance()	dx_make('secure')
`cache`	'DxCache' (static 클래스)	dx_make('cache')
`hook` / `hooks`	HookManager::getInstance()	dx_make('hook')
`seo`	'DxSeo' (static 메서드)	dx_make('seo')
`site`	DxSite::getInstance()	dx_make('site')
`theme`	DxTheme::getInstance()	dx_make('theme')

5. HookManager — 훅 시스템

WordPress의 add_action/apply_filters와 동일한 철학으로 구현된 이벤트 기반 확장 시스템입니다. 플러그인이나 extend/ 파일이 CMS 코어를 직접 수정하지 않고 실행 흐름에 끼어들 수 있도록 합니다.

5.1 Action Hook vs Filter Hook

구분	메서드	동작	반환값
Action Hook	dx_add_hook / dx_run_hook	특정 시점에 코드 실행	없음 (void)
Filter Hook	dx_add_filter / dx_apply_filter	값을 받아 변형 후 반환	변형된 값 반환

5.2 훅 등록 및 실행

```
// Action 훅 등록 (우선순위 10, 낮을수록 먼저 실행)
dx_add_hook('dx_head', function() {
    echo '<link rel="stylesheet" href="/custom.css">';
}, 10);

// Action 훅 실행 (코어에서 호출)
dx_run_hook('dx_head');  // → 등록된 모든 콜백을 우선순위 순으로 실행

// Filter 훅 등록 (게시글 내용 변형)
dx_add_filter('dx_board_content', function($content, $args) {
    return nl2br($content);  // 줄바꿈 → <br> 변환
}, 20);

// Filter 훅 실행 (코어에서 호출)
$content = dx_apply_filter('dx_board_content', $rawContent, $ctx);
```

5.3 페이지별 자동 훅 포인트

각 페이지 렌더링 시 dx_hook_top(), dx_hook_middle(), dx_hook_bottom() 이 자동 호출됩니다. 이 함수들은 전역 훅과 페이지 타입별 훅, 슬러그별 훅을 체계적으로 실행합니다.

```
// dx_hook_top($context) 내부 동작
```

```php
dx_run_hook('dx_top', $context);            // 모든 페이지 공통
dx_run_hook('dx_board_top', $context);      // 게시판 타입만
($context['type']='board')
dx_run_hook('dx_page_about_top', $ctx);     // 특정 슬러그만
($context['slug']='about')
```

5.4 주요 훅 포인트 전체 목록

훅 이름	실행 시점	주요 활용
dx_head	<head> 태그 내부	CSS/JS 추가, meta 태그 삽입
dx_body_top	<body> 직후	배너, 공지 삽입
dx_body_bottom	</body> 직전	GA 코드, 팝업, 채팅 위젯 삽입
dx_top	페이지 본문 최상단	점검 모드, IP 차단
dx_middle	페이지 본문 중간	A/B 테스트, 추가 위젯
dx_bottom	페이지 본문 최하단	성능 측정, 트래킹
dx_board_list_context	게시판 목록 컨텍스트 생성 후	컨텍스트 변수 추가/변경
dx_board_view_context	게시글 보기 컨텍스트 생성 후	추가 데이터 주입
dx_board_after_save	게시글 저장 완료 후	알림 발송, 포인트 적립, 인덱싱
dx_after_login	로그인 완료 후	접속 로그, 알림 처리
dx_after_logout	로그아웃 완료 후	세션 정리, 로그 기록
dx_editor_render	에디터 HTML 렌더링 시	에디터 설정 변경
dx_admin_top	관리자 본문 상단	관리자 커스텀 메뉴 추가

6. Secure — 보안 엔진

Secure.php는 CMS의 모든 보안 코드를 하나의 파일에 집중시킨 전담 클래스입니다. 보안 패치가 필요할 때 이 파일 하나만 교체하면 됩니다. v5.2.2에서 WAF, Rate Limit, Bot 탐지 기능이 추가되었습니다.

Secure.php 위치 난독화

- 설치 시 16자리 랜덤 해시(DX_SECURITY_PATH)를 생성합니다
- Secure.php의 실제 경로: core/security/{16자리해시}/Secure.php
- 소스코드가 유출되어도 Secure.php의 경로를 예측할 수 없습니다
- index.php는 config.php에서 해시를 읽어 동적으로 경로를 조합합니다

6.1 담당 기능 전체

기능	구현 방식	설명
세션 보안	initSession() + startSession()	HttpOnly · Secure · SameSite=Lax 쿠키 플래그
CSRF 방어	csrfToken() / csrfCheck()	TTL 3시간, 토큰 불일치 시 403 반환
보안 헤더	sendSecurityHeaders()	X-Frame-Options · X-Content-Type-Options · Referrer-Policy · CSP Nonce
XSS 방어	esc() / sanitize()	출력 이스케이프, HTML 입력 정제
WAF	wafRules 배열 + 검사 로직	SQL Injection · XSS 패턴 탐지, POST 에디터 필드 제외
Rate Limiting	rateLimit()	Redis 기반 (파일 fallback), 10초 내 60회 / IP 별 분당 200회
Bot 탐지	allowedBots 화이트리스트	차단 아닌 로그만 기록 (검색엔진 봇 보호)
비밀번호 해시	bcryptHash() / bcryptVerify()	bcrypt, PHP 5.6 fallback 포함
파일 업로드 검증	validateUpload()	MIME + 확장자 이중 검증, 이중 확장자 공격 차단

기능	구현 방식	설명
경로 순회 방어	safeUrl()	Path Traversal 공격 입력값 차단
안전 난수	randomBytes() / randomHex()	PHP 5.6 ~ 8.x 호환 안전 난수 생성
IP 추출	clientIp()	Cloudflare · 리버스프록시 · X-Forwarded-For 처리

6.2 CSRF 방어 흐름

```
// 1. 토큰 발급 (GET 요청 시, 세션 있을 때)
$token = Secure::csrfToken();    // 세션에 저장, TTL 3시간


// 2. 폼에 삽입
echo Secure::csrfField();        // <input type="hidden" name="_csrf" value="...">
// 또는 전역 함수 사용
echo dx_csrf_field();            // 동일한 결과


// 3. POST 요청 검증
if (!Secure::csrfCheck()) {      // POST 요청에서 토큰 검증
    http_response_code(403);
    exit('CSRF 토큰 불일치');
}
// 또는 미들웨어 방식 (DxRouter)
DxRouter::post('/submit', 'Ctrl@handle')->middleware('csrf');
```

6.3 WAF (웹 방화벽) 동작

WAF 적용 범위 및 제외

- 검사 대상: GET 파라미터, POST 파라미터 (단, 에디터 필드 제외)
- 제외 필드: content, body, description, editor_content, comment 등 — HTML 입력을 허용하는 필드는 WAF 오탐 방지를 위해 제외
- SQL Injection 탐지: UNION SELECT, information_schema, sleep() 등
- XSS 탐지: <script>, javascript:, onerror=, document.cookie 등
- 탐지 시 동작: 요청 차단 + 로그 기록 (data/error.log)

7. Database + QueryBuilder — DB 레이어

Database는 PDO 래퍼 싱글턴이고, QueryBuilder는 라라벨 스타일의 메서드 체이닝 쿼리 빌더입니다. 두 클래스는 함께 사용할 수 있으며, 테이블 prefix 처리가 자동화되어 있습니다.

7.1 Database 클래스 주요 메서드

메서드	설명	반환값
connect()	PDO 연결 초기화 (config.php에서 호출)	$this (체이닝)
pdo()	PDO 인스턴스 직접 반환 (고급 쿼리용)	PDO 객체
row($sql, $params)	단일 행 SELECT	배열 or null
rows($sql, $params)	전체 행 SELECT	배열[]
execute($sql, $params)	INSERT/UPDATE/DELETE 실행	PDOStatement
insert($table, $data)	INSERT (prefix 자동 처리)	lastInsertId
update($table, $data, $where)	UPDATE (prefix 자동 처리)	영향받은 행 수
delete($table, $where)	DELETE (prefix 자동 처리)	영향받은 행 수
find($table, $where)	단일 행 찾기	배열 or null
count($table, $where)	COUNT 쿼리	int
table($name)	prefix 붙인 테이블명 반환	문자열 (예: dx_members)
beginTransaction()	트랜잭션 시작	void
commit() / rollback()	트랜잭션 커밋/롤백	void

7.2 QueryBuilder 사용법

```
// dx_db() 전역 함수로 접근
$posts = dx_db('posts')              // dx_posts 테이블
    ->select('id, title, created_at')
    ->where('status', 1)
```

```
    ->where('board_key', 'free')
    ->orderBy('created_at', 'DESC')
    ->limit(10)
    ->offset(0)
    ->get();                          // 실행 → 배열[]

// 단일 행
$member = dx_db('members')->where('id', 1)->first();

// INSERT
$id = dx_db('posts')->insert(array('title'=>'제목', 'content'=>'내용'));

// UPDATE
dx_db('posts')->where('id', 5)->update(array('title'=>'수정된 제목'));

// WHERE IN + LIKE
dx_db('posts')->whereIn('id', array(1,2,3))->get();
dx_db('posts')->whereLike('title', '%검색어%')->get();
```

8. DxCache — 멀티 드라이버 캐시

DxCache는 Redis → APCu → 파일 캐시 → None 순서로 드라이버를 자동 선택합니다. 어떤 환경에서도 동일한 API로 동작하며, 저가형 공유호스팅도 파일 캐시로 지원합니다.

8.1 드라이버 선택 로직

```
// DxCache::init() 내부 — 최초 1회 실행

// 1순위: Redis (REDIS_SESSION_URL 설정 + Redis 익스텐션 + 연결 성공)
if (Secure::getRedis() !== null) {
    self::$driver = 'redis';  // 원자적 연산, 다중 서버 공유 가능
}
// 2순위: APCu (apc.enabled + apcu_fetch 존재)
else if (function_exists('apcu_fetch') && ini_get('apc.enabled')) {
    self::$driver = 'apcu';   // PHP-FPM 프로세스 공유 메모리
}
// 3순위: 파일 캐시 (data/cache/ 쓰기 가능)
else if (is_writable(DX_DATA.'/cache')) {
    self::$driver = 'file';   // 원자적 쓰기 (tmp → rename)
}
// 4순위: None (캐시 없이 동작)
else { self::$driver = 'none'; }
```

8.2 캐시 항목과 TTL

캐시 키	TTL	무효화 시점
dx_settings (전체 설정)	5분	관리자 설정 저장 시 전체 flush
board_{key}_list (게시판 목록)	1분	게시글 작성/수정/삭제 시 해당 게시판 캐시만 삭제
sitemap_xml	10분	게시글 변경 시
category_tree	5분	카테고리 변경 시 전체 게시판 목록 캐시도 삭제

8.3 캐시 사용법

```
// 저장
DxCache::set('my_key', $data, 300);  // TTL 300초

// 읽기
$data = DxCache::get('my_key');       // 없으면 null

// 삭제
DxCache::delete('my_key');
DxCache::flush();                     // 전체 삭제

// 현재 드라이버 확인
$driver = DxCache::driver();          // 'redis' | 'apcu' | 'file' | 'none'
```

9. Auth — 세션 기반 인증

Auth는 세션 기반의 인증 싱글턴입니다. DX_SECRET_KEY 기반 동적 세션 키를 사용하며, Remember Me 쿠키를 통한 자동 로그인도 지원합니다.

9.1 인증 흐름

```
// 1. 세션에서 사용자 정보 로드 (Auth 생성 시 자동 실행)
//    세션 키: substr(sha1('dx_user' + DX_SECRET_KEY), 0, 12)
$sessionData = $_SESSION[$this->sessionKey()];

// 2. DB에서 사용자 검증 (status=1, 토큰 일치)
$user = $db->find('members', array('id'=>$userId, 'status'=>1));
if ($user && $sessionData['token'] === $this->makeToken($user)) {
    $this->user = $user;  // 인증 성공
} else {
    $this->tryRememberMe();  // Remember Me 쿠키 시도
}

// 3. 사용법
$auth = Auth::getInstance();
$auth->isLoggedIn();           // 로그인 여부
$auth->isAdmin();              // 관리자 여부
$auth->get('id');              // 사용자 ID
$auth->get('nickname');        // 닉네임
$auth->login($userId);         // 로그인 처리
$auth->logout();               // 로그아웃
```

9.2 소셜 로그인 (DxSocialAuth)

카카오, 네이버, 구글, GitHub의 OAuth2 인증을 DxSocialAuth 클래스가 처리합니다. OAuth 콜백 → 사용자 정보 조회 → dx_social_accounts 테이블 매핑 → 세션 로그인의 흐름으로 동작합니다.

제공자	인증 방식	필요 설정
카카오	OAuth2 (REST API 키)	관리자 > 소셜 설정 > 카카오 앱 키
네이버	OAuth2	관리자 > 소셜 설정 > 네이버 클라이언트 ID/Secret
구글	OAuth2	관리자 > 소셜 설정 > 구글 클라이언트 ID/Secret

제공자	인증 방식	필요 설정
GitHub	OAuth2	관리자 > 소셜 설정 > GitHub 클라이언트 ID/Secret

10. DxTheme — 테마 엔진

DxTheme은 폴백 체인을 갖춘 테마 렌더링 엔진입니다. 커스텀 테마에 파일이 없으면 자동으로 default 테마 파일을 사용합니다.

10.1 폴백 체인

```
// 예: 현재 테마가 'my-theme'이고 board/list.php를 찾을 때

// 1순위: 현재 테마
themes/my-theme/board/list.php       ← 있으면 사용

// 2순위: default 테마 (폴백)
themes/default/board/list.php        ← 없으면 여기 사용

// 3순위: 404 (두 경우 모두 없을 때)
themes/default/page/404.php
```

10.2 레이아웃 렌더링 흐름

```
// Dispatcher가 컨텐츠를 출력하면 DxTheme이 레이아웃으로 감쌈

// 1. 컨텐츠 출력 버퍼링 시작
ob_start();
// 2. 게시판 핸들러 실행 (boards/handler.php)
require $handlerFile;
$content = ob_get_clean();    // 컨텐츠 캡처

// 3. 레이아웃에 컨텐츠 주입
// themes/default/layout/main.php 내부:
// <?php include DxTheme::resolve('layout/header.php'); ?>
// <main><?php echo $content; ?></main>
// <?php include DxTheme::resolve('layout/footer.php'); ?>
```

10.3 테마 파일 resolve 메서드

```php
// 파일 경로 확정 (폴백 포함)
$path = DxTheme::resolve('board/list.php');
// 반환: themes/my-theme/board/list.php (존재 시)
//       themes/default/board/list.php  (폴백)

// 파일 렌더링 (변수 주입)
DxTheme::render('board/list.php', array('posts'=>$posts, 'board'=>$board));
// → 파일에서 $posts, $board 변수로 접근 가능
```

11. DxSite — 멀티사이트 엔진

DxSite는 단일 CMS 설치로 여러 도메인의 사이트를 운영할 수 있게 해주는 멀티사이트 관리자입니다. HTTP_HOST를 감지하여 dx_sites 테이블에서 도메인별 설정을 로드하고 전역 $dx_config를 오버라이드합니다.

11.1 도메인 설정 로드 흐름

```
// DxSite::__construct() 내부 흐름
$domain = $_SERVER['HTTP_HOST'];  // 포트 번호 제거 후 소문자화

// dx_sites 테이블에서 도메인 조회
$site = $db->find('sites', array('domain'=>$domain, 'status'=>1));

if ($site) {
    // 전역 설정 오버라이드
    $dx_config['site_name'] = $site['site_name'];
    $dx_config['theme']     = $site['theme'];
    $dx_config['menu_group']= $site['menu_group'];
    $dx_config['timezone']  = $site['timezone'];
    // ... 언어, SEO 설정 등도 오버라이드
}
// 미등록 도메인이면 dx_settings 기본값 그대로 사용
```

12. DxExtend — 코드 자동 삽입 시스템

DxExtend는 훅 등록 없이 파일만 특정 폴더에 넣으면 CMS가 자동으로 실행해주는 코드 삽입 시스템입니다. 파일명 오름차순으로 실행되며, 개별 파일 오류가 다른 파일에 영향을 주지 않도록 격리됩니다.

폴더	실행 함수	실행 시점	특징
extend/top/	runTop()	STEP 4 완료 직후	모든 서비스 초기화 완료 상태
extend/middle/	runMiddle()	라우팅 결정 후, 컨트롤러 실행 전	URL·사용자 정보 접근 가능

폴더	실행 함수	실행 시점	특징
extend/bottom/	runBottom()	응답 출력 완료 후	출력 버퍼 후처리 가능

```php
// DxExtend::runTop() 내부 동작
$files = glob(DX_EXTEND . '/top/*.php');
sort($files);  // 파일명 오름차순 (01_ 접두사로 순서 제어)
foreach ($files as $file) {
    try { require $file; }   // 각 파일 독립 실행
    catch (Exception $e) {  // 오류 격리 — 다른 파일에 영향 없음
        dx_error_log($e->getMessage());
    }
}
```

13. PluginRegistry — 플러그인 시스템

PluginRegistry는 plugins/ 폴더를 스캔하여 활성화된 플러그인의 plugin.php를 로드하고, dx_register_plugin()으로 등록된 플러그인 메타데이터를 관리하는 레지스트리입니다.

13.1 플러그인 로드 순서

```
// load_plugins() 내부 흐름 (STEP 4)

// 1. DB에서 활성 플러그인 목록 조회
$activePlugins = $db->rows('SELECT id FROM dx_plugins WHERE active=1');

// 2. 각 플러그인 폴더의 plugin.php 로드
foreach ($activePlugins as $plugin) {
    $file = DX_PLUGINS . '/' . $plugin['id'] . '/plugin.php';
    if (file_exists($file)) require $file;   // 훅 등록 등 실행
}
```

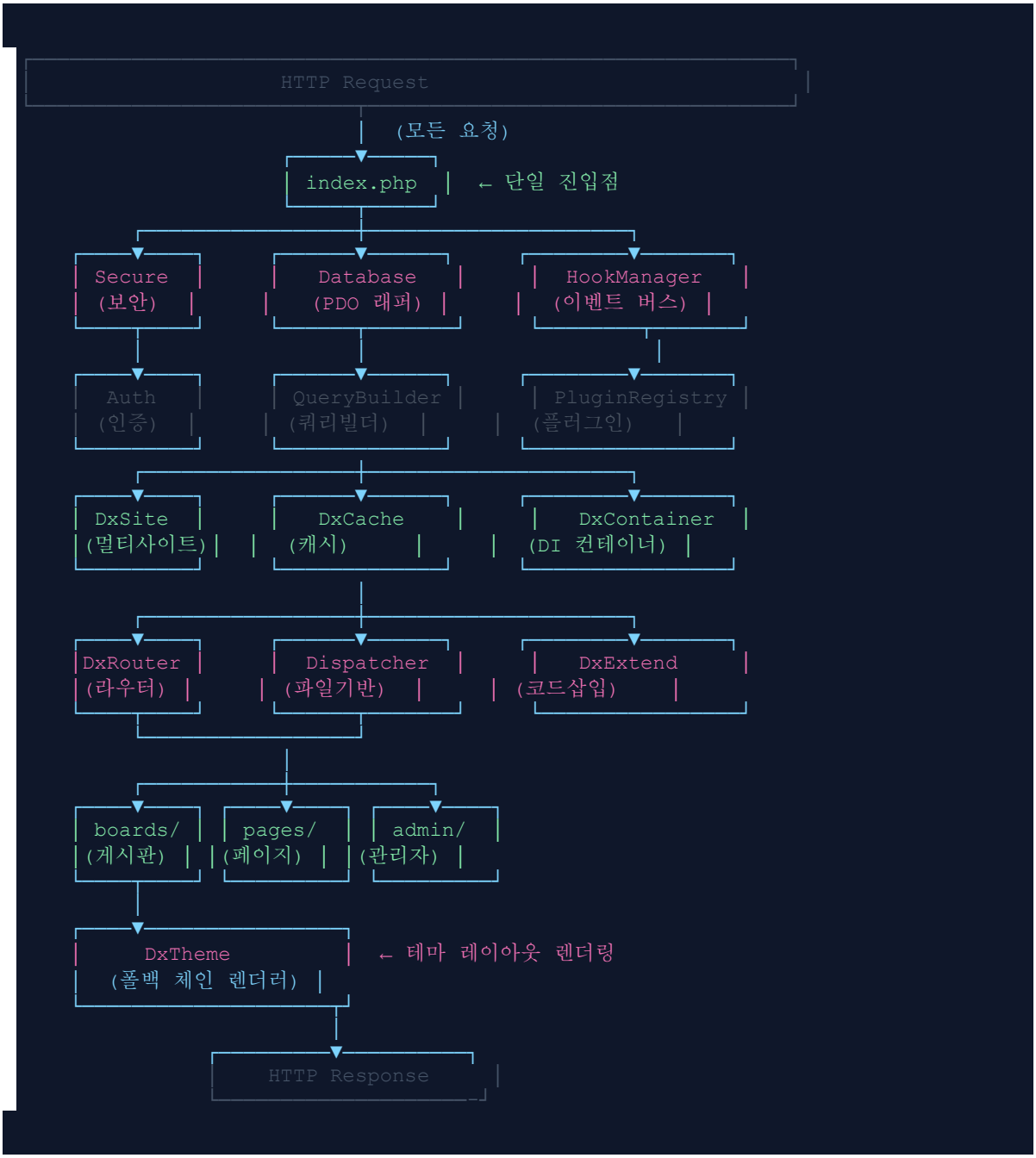
13.2 플러그인 등록 API

```
// plugins/my-plugin/plugin.php
dx_register_plugin(array(
    'id'      => 'my-plugin',
    'type'    => 'editor',   // editor | payment | socket | captcha
    'name'    => 'My Plugin',
    'version' => '1.0.0',
    'author'  => 'My Name',
    'settings' => array(
        'my_option' => array(
            'label'   => '옵션명',
            'type'    => 'select',  // text | select | checkbox | textarea
            'options' => array('1'=>'사용','0'=>'사용 안 함'),
            'default' => '1',
        ),
    ),
));

// 훅으로 기능 연결
dx_add_hook('dx_editor_render', function($args) {
    // 에디터 HTML 출력
}, 10);
```

14. 전체 엔진 아키텍처 요약

아래는 DXCMS 미니 프레임워크의 요청 처리 전체 아키텍처를 계층별로 정리한 것입니다.



14.1 엔진 설계 원칙 요약

원칙	구현 방법	효과
단일 진입점	index.php Front Controller	URL 처리 중앙화, 보안 일관성 유지
관심사 분리	Secure, Auth, Cache, Router 클래스 분리	각 모듈 독립 교체/패치 가능
폴백 보장	Redis→APCu→파일, 테마→default	어떤 환경에서도 항상 동작
PHP 5.6 호환	?? 연산자, 타입힌트 없이 구현	공유호스팅 포함 전 환경 지원
라라벨 스타일 API	DxRouter, DxContainer, QueryBuilder	PHP 개발자에게 익숙한 인터페이스
이벤트 기반 확장	HookManager (Action + Filter)	코어 수정 없이 플러그인으로 확장
보안 집중화	Secure.php 하나에 모든 보안 코드	보안 패치 시 파일 1개만 교체
비로그인 최적화	GET 비로그인 요청은 세션 시작 안 함	파일 락 제거, 동시 처리 성능 향상