1 PIANO REINGEGNERIZZAZIONE ENTERPRISE - DEALERCLOUD

1.1 🎯 OBIETTIVI STRATEGICI

1.2 📅 FASE 1: FONDAMENTA SICUREZZA

1.2.1 1.1 SICUREZZA DATABASE

1.2.2 1.2 SICUREZZA WEB APPLICATION

1.2.3 1.3 SESSION & AUTHENTICATION

1.3 📅 FASE 2: ARCHITETTURA MODERNA

1.3.1 2.1 REFACTORING ARCHITETTURALE

1.3.2 2.2 PERFORMANCE OPTIMIZATION

1.4 📅 FASE 3: QUALITÀ & TESTING

1.4.1 3.1 TESTING ENTERPRISE

1.4.2 3.2 CODE QUALITY

1.5 📅 FASE 4: INFRASTRUTTURA ENTERPRISE

1.5.1 4.1 MONITORING & LOGGING

1.5.2 4.2 DEPLOYMENT & CI/CD

1.5.3 4.3 DISASTER RECOVERY & BACKUP

1.6 📅 FASE 5: FEATURES ENTERPRISE

1.6.1 5.1 MULTI-TENANCY (se necessario)

1.6.2 5.2 API GATEWAY

1.6.3 5.3 ADVANCED FEATURES

1.7 📊 METRICHE SUCCESSO

1.7.1 Sicurezza

1.7.2 Performance

1.7.3 Qualità

1.7.4 Affidabilità

1.8 ⚠️ RISCHI E MITIGAZIONE

1.8.1 Rischio 1: Downtime durante migrazione

1.8.2 Rischio 2: Regressioni funzionali

1.8.3 Rischio 3: Performance degradate

1.8.4 Rischio 4: Budget overrun

1.9 📋 CHECKLIST IMPLEMENTAZIONE

1.9.1 Fase 1 - Sicurezza

1.9.2 Fase 2 - Architettura

1.9.3 Fase 3 - Qualità

1.9.4 Fase 4 - Infrastruttura

# 1 PIANO REINGEGNERIZZAZIONE ENTERPRISE - DEALERCLOUD

**Data:** 2025-11-24
**Obiettivo:** Trasformare DealerCloud da sistema legacy a piattaforma enterprise-grade

---

## 1.1 🎯 OBIETTIVI STRATEGICI

1. **Sicurezza Enterprise:** Eliminare tutte le vulnerabilità critiche
2. **Performance:** Migliorare throughput del 300-500%
3. **Scalabilità:** Supportare 10,000+ impianti senza degradazione
4. **Manutenibilità:** Codice moderno, testabile, documentato
5. **Affidabilità:** 99.9% uptime, disaster recovery
6. **Compliance:** GDPR, PCI-DSS (se applicabile), audit trail completo

---

## 1.2 📅 FASE 1: FONDAMENTA SICUREZZA

### 1.2.1 1.1 SICUREZZA DATABASE

#### 1.2.1.1 Task 1.1.1: Prepared Statements Migration

**Priorità:** CRITICA

**Obiettivo:** Convertire tutte le 374 query SQL a prepared statements

**Approccio:** 1. Analisi completa query esistenti 2. Creazione wrapper `Database` class con prepared statements 3. Migrazione query critiche (login, API) per prime 4. Migrazione query rimanenti 5. Testing completo

**Codice Target:**

```
// PRIMA (VULNERABILE)
$query = "SELECT * FROM utente WHERE email = '$usr' AND password = '$psw'";
$result = sqlsrv_query($conn, $query);

// DOPO (SICURO)
$stmt = $db->prepare("SELECT * FROM utente WHERE email = ? AND password =
        ?");
$stmt->bind_param("ss", $usr, $psw);
$result = $stmt->execute();
```

**Deliverable:** - Database class con prepared statements - Tutte le 374 query migrate - Test suite sicurezza - Documentazione migrazione

**Success Criteria:** - 0 query con SQL injection possibile - Tutti i test sicurezza passati - Performance mantenute o migliorate

---

#### 1.2.1.2 Task 1.1.2: Credenziali Management

**Priorità:** CRITICA

**Obiettivo:** Centralizzare credenziali in file sicuro esterno

**Approccio:** 1. Creare `/opt/dealercloud/config/db.ini` (fuori webroot) 2. Implementare `Config` class per caricamento sicuro 3. Migrare tutti i `cn.php` a usare Config class 4. Rimuovere credenziali hardcoded 5. Implementare rotazione password

**Struttura:**

```
/opt/dealercloud/
├── config/
|   ├── db.ini (600, root:root)
|   ├── smtp.ini
|   └── api.ini
└── wwwroot/
    └── dealercloud/ (nessuna credenziale qui)
```

**Deliverable:** - Config class centralizzata - File configurazione sicuri - Documentazione gestione credenziali - Script rotazione password

---

#### 1.2.1.3 Task 1.1.3: Password Hashing Enterprise

**Priorità:** CRITICA

**Obiettivo:** Implementare password hashing sicuro (Argon2id)

**Approccio:** 1. Creare `Password` utility class 2. Migrare login a password_verify() 3. Script migrazione password esistenti (hash al primo login) 4. Implementare password policy (min 12 caratteri, complessità) 5. Implementare password reset sicuro

**Codice:**

```
class Password {
    public static function hash($password) {
        return password_hash($password, PASSWORD_ARGON2ID, [
            'memory_cost' => 65536,
            'time_cost' => 4,
            'threads' => 3
        ]);
    }

    public static function verify($password, $hash) {
        return password_verify($password, $hash);
    }
}
```

**Deliverable:** - Password utility class - Login migrato - Script migrazione password - Password policy implementata

---

## 1.2.2 1.2 SICUREZZA WEB APPLICATION

### 1.2.2.1 Task 1.2.1: CSRF Protection

**Priorità:** ALTA

**Obiettivo:** Proteggere tutti i form e API da CSRF

**Approccio:** 1. Implementare CSRF token generator/validator 2. Middleware CSRF per tutte le richieste POST/PUT/DELETE 3. Integrare token in tutti i form (100+ form) 4. Validazione token su tutte le API 5. Testing completo

**Codice:**

```php
class CSRF {
    public static function generate() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }

    public static function validate($token) {
        return isset($_SESSION['csrf_token']) &&
               hash_equals($_SESSION['csrf_token'], $token);
    }
}
```

**Deliverable:** - CSRF protection completo - Tutti i form protetti - API protette - Test suite CSRF

---

#### 1.2.2.2 Task 1.2.2: XSS Prevention

**Priorità:** ALTA

**Obiettivo:** Sanitizzare tutti gli output

**Approccio:** 1. Creare `Output` utility class per escape 2. Audit completo output non sanitizzato (30+ file) 3. Migrazione output a escape automatico 4. Content Security Policy (CSP) headers 5. Testing XSS completo

**Codice:**

```
class Output {
    public static function escape($string, $context = 'html') {
        switch ($context) {
            case 'html':
                return htmlspecialchars($string, ENT_QUOTES | ENT_HTML5,
        'UTF-8');
            case 'attr':
                return htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
            case 'js':
                return json_encode($string, JSON_HEX_TAG | JSON_HEX_AMP |
        JSON_HEX_APOS | JSON_HEX_QUOT);
            default:
                return $string;
        }
    }
}
```

**Deliverable:** - Output utility class - Tutti gli output sanitizzati - CSP headers configurati - Test suite XSS

---

#### 1.2.2.3 Task 1.2.3: File Upload Security

**Priorità:** ALTA

**Obiettivo:** Validare e proteggere upload file

**Approccio:** 1. Whitelist estensioni file permesse 2. Validazione MIME type (non solo estensione) 3. Scanning antivirus file uploadati 4. Storage file fuori webroot 5. Rate limiting upload per utente

**Deliverable:** - File upload sicuro - Validazione completa - Storage sicuro - Rate limiting

---

### 1.2.3 1.3 SESSION & AUTHENTICATION

#### 1.2.3.1 Task 1.3.1: Session Security Hardening

**Priorità:** ALTA

**Obiettivo:** Rafforzare sicurezza sessioni

**Approccio:** 1. Session ID rotation ogni richiesta critica 2. Validazione IP address (opzionale, configurabile) 3. Timeout sessioni più brevi (15 minuti) 4. Invalidazione sessioni multiple per utente 5. Secure flag sempre attivo

**Deliverable:** - Session management sicuro - Configurazione flessibile - Documentazione

---

#### 1.2.3.2 Task 1.3.2: Rate Limiting & Brute Force Protection

**Priorità:** ALTA

**Obiettivo:** Proteggere login e API da abusi

**Approccio:** 1. Implementare rate limiting (Redis/Memcached) 2. Login: max 5 tentativi per IP/15 minuti 3. API: max 100 richieste/minuto per API key 4. Account lockout dopo tentativi falliti 5. Logging tentativi falliti

**Deliverable:** - Rate limiting implementato - Brute force protection - Monitoring e alerting

---

## 1.3 📅 FASE 2: ARCHITETTURA MODERNA

### 1.3.1 2.1 REFACTORING ARCHITETTURALE

#### 1.3.1.1 Task 2.1.1: Introduzione Framework MVC

**Priorità:** ALTA

**Obiettivo:** Migrare a architettura MVC moderna

**Scelta Framework:** Laravel 11 o Symfony 6 (raccomandato Laravel per rapidità)

**Approccio:** 1. Setup Laravel/Symfony in parallelo 2. Migrazione moduli critici per primi: - Authentication - API endpoints - Dashboard principale 3. Migrazione moduli rimanenti 4. Testing completo

**Architettura Target:**

```
dealercloud/
├── app/
|   ├── Http/
|   |   ├── Controllers/
|   |   |   ├── AuthController.php
|   |   |   ├── Api/
|   |   |   |   ├── FatturaController.php
|   |   |   |   └── ImpiantiController.php
|   |   |   └── DashboardController.php
|   |   ├── Middleware/
|   |   |   ├── Authenticate.php
|   |   |   ├── CSRF.php
|   |   |   └── RateLimit.php
|   |   └── Requests/
|   |       └── ApiRequest.php
|   ├── Models/
|   |   ├── User.php
|   |   ├── Impianto.php
|   |   └── Fattura.php
|   ├── Services/
|   |   ├── AuthService.php
|   |   └── ApiService.php
|   └── Repositories/
|       ├── UserRepository.php
|       └── ImpiantoRepository.php
├── config/
├── database/
|   └── migrations/
├── routes/
|   ├── web.php
|   └── api.php
└── tests/
```

**Deliverable:** - Framework MVC implementato - Moduli critici migrati - Architettura pulita - Test suite completa

---

#### 1.3.1.2 Task 2.1.2: Database Abstraction Layer

**Priorità:** ALTA

**Obiettivo:** Implementare ORM/Query Builder moderno

**Approccio:** 1. Utilizzare Eloquent (Laravel) o Doctrine (Symfony) 2. Creare Models per tutte le tabelle principali 3. Migrare query a ORM 4. Implementare Repository pattern 5. Testing performance

**Codice Target:**

```
// PRIMA
$query = "SELECT * FROM Impianto WHERE IdClienteAlfa = $id";
$result = sqlsrv_query($conn, $query);

// DOPO
$impianti = Impianto::where('IdClienteAlfa', $id)->get();
```

**Deliverable:** - ORM implementato - Models completi - Repository pattern - Performance ottimizzate

---

#### 1.3.1.3 Task 2.1.3: API RESTful Moderna

**Priorità:** ALTA

**Obiettivo:** Refactoring API a standard RESTful

**Approccio:** 1. Design API RESTful completo 2. Implementazione con Laravel API Resources 3. Versioning API (v1, v2) 4. Documentazione OpenAPI/Swagger 5. Testing API completo

**API Design:**

```
GET    /api/v1/impianti              # Lista impianti
GET    /api/v1/impianti/{id}         # Dettaglio impianto
POST   /api/v1/impianti              # Crea impianto
PUT    /api/v1/impianti/{id}         # Aggiorna impianto
DELETE /api/v1/impianti/{id}         # Elimina impianto

GET    /api/v1/impianti/{id}/fatture # Fatture impianto
GET    /api/v1/fatture/{id}          # Dettaglio fattura
```

**Deliverable:** - API RESTful completa - Documentazione Swagger - Versioning API - Test suite API

---

### 1.3.2 2.2 PERFORMANCE OPTIMIZATION

#### 1.3.2.1 Task 2.2.1: Database Optimization

**Priorità:** ALTA

**Obiettivo:** Ottimizzare performance database

**Approccio:** 1. Analisi query plan execution 2. Creazione indici mancanti (132 indici) 3. Rimozione indici non utilizzati (116 indici) 4. Deframmentazione indici (22 indici) 5. Ottimizzazione query pesanti 6. Implementazione query caching

**Query Optimization:** - Sostituire `SELECT *` con colonne specifiche (2034 query) - Fixare N+1 queries con JOIN (3175 loop) - Implementare paginazione su tutte le liste - Aggiungere indici compositi per query frequenti

**Deliverable:** - Database ottimizzato - Indici corretti - Query performance migliorate - Monitoring performance

---

#### 1.3.2.2 Task 2.2.2: Caching Strategy

**Priorità:** ALTA

**Obiettivo:** Implementare caching multi-layer

**Approccio:** 1. Redis setup e configurazione 2. Query caching (query frequenti) 3. Application caching (dati statici) 4. Session caching (Redis) 5. CDN per asset statici

**Caching Layers:** - **L1:** Application cache (APCu/Memcached) - dati in-memory - **L2:** Redis cache - dati condivisi tra istanze - **L3:** Database query cache - query results - **L4:** CDN - asset statici (JS, CSS, immagini)

**Deliverable:** - Caching implementato - Performance migliorate 3-5x - Monitoring cache hit rate

---

#### 1.3.2.3 Task 2.2.3: Connection Pooling

**Priorità:** MEDIA

**Obiettivo:** Implementare connection pooling

**Approccio:** 1. Configurare SQL Server connection pooling 2. Implementare PDO connection pool 3. Monitoring connessioni attive 4. Tuning pool size

**Deliverable:** - Connection pooling attivo - Scalabilità migliorata - Monitoring connessioni

---

# 1.4 📅 FASE 3: QUALITÀ & TESTING

## 1.4.1 3.1 TESTING ENTERPRISE

### 1.4.1.1 Task 3.1.1: Unit Testing

**Priorità:** ALTA

**Obiettivo:** Coverage testing >80%

**Approccio:** 1. Setup PHPUnit 2. Test critici per primi (auth, API, business logic) 3. Test coverage analysis 4. CI/CD integration 5. Test automation

**Deliverable:** - Test suite completa - Coverage >80% - CI/CD pipeline - Test automation

---

### 1.4.1.2 Task 3.1.2: Integration Testing

**Priorità:** ALTA

**Obiettivo:** Test integrazione end-to-end

**Approccio:** 1. Test database integration 2. Test API integration 3. Test authentication flow 4. Test business workflows

**Deliverable:** - Integration tests - E2E tests - Test reports

---

### 1.4.1.3 Task 3.1.3: Security Testing

**Priorità:** CRITICA

**Obiettivo:** Penetration testing completo

**Approccio:** 1. Automated security scanning (OWASP ZAP, Burp Suite) 2. Manual penetration testing 3. Code review sicurezza 4. Vulnerability assessment 5. Remediation

**Deliverable:** - Security audit report - Vulnerabilità risolte - Security best practices documentate

---

## 1.4.2 3.2 CODE QUALITY

### 1.4.2.1 Task 3.2.1: Code Standards & Linting

**Priorità:** MEDIA

**Obiettivo:** Standardizzare codice

**Approccio:** 1. PSR-12 coding standards 2. PHPStan static analysis 3. PHP_CodeSniffer 4. Pre-commit hooks 5. Code review process

**Deliverable:** - Code standards documentati - Linting automatizzato - Code review process

---

#### 1.4.2.2 Task 3.2.2: Documentation

**Priorità:** MEDIA

**Obiettivo:** Documentazione completa

**Approccio:** 1. API documentation (Swagger) 2. Code documentation (PHPDoc) 3. Database schema documentation 4. Deployment documentation 5. User manual

**Deliverable:** - Documentazione completa - API docs - Developer guide - User manual

---

## 1.5 📅 FASE 4: INFRASTRUTTURA ENTERPRISE

### 1.5.1 4.1 MONITORING & LOGGING

#### 1.5.1.1 Task 4.1.1: Application Monitoring

**Priorità:** ALTA

**Obiettivo:** Monitoring completo applicazione

**Approccio:** 1. APM (Application Performance Monitoring) - New Relic/DataDog 2. Error tracking - Sentry 3. Log aggregation - ELK Stack 4. Metrics dashboard - Grafana 5. Alerting configurazione

**Deliverable:** - Monitoring completo - Dashboards - Alerting configurato

---

#### 1.5.1.2 Task 4.1.2: Structured Logging

**Priorità:** ALTA

**Obiettivo:** Logging strutturato per audit

**Approccio:** 1. PSR-3 logging 2. Structured logs (JSON) 3. Log levels corretti 4. Audit trail completo 5. Log retention policy

**Deliverable:** - Logging strutturato - Audit trail - Log retention

---

## 1.5.2 4.2 DEPLOYMENT & CI/CD

### 1.5.2.1 Task 4.2.1: CI/CD Pipeline

**Priorità:** ALTA

**Obiettivo:** Deployment automatizzato

**Approccio:** 1. GitLab CI / GitHub Actions 2. Automated testing 3. Automated deployment 4. Blue-green deployment 5. Rollback automatizzato

**Pipeline:**

```
Commit → Test → Build → Deploy Staging → Test E2E → Deploy Production
```

**Deliverable:** - CI/CD pipeline - Automated deployment - Rollback capability

---

### 1.5.2.2 Task 4.2.2: Infrastructure as Code

**Priorità:** MEDIA

**Obiettivo:** Infrastruttura versionabile

**Approccio:** 1. Docker containers 2. Kubernetes (se necessario) 3. Terraform per infrastruttura 4. Ansible per configurazione

**Deliverable:** - Infrastructure as Code - Containers - Orchestration

---

## 1.5.3 4.3 DISASTER RECOVERY & BACKUP

### 1.5.3.1 Task 4.3.1: Backup Strategy

**Priorità:** ALTA

**Obiettivo:** Backup automatizzato e testato

**Approccio:** 1. Database backup giornaliero 2. File backup incrementale 3. Backup offsite 4. Test restore regolari 5. RTO/RPO definiti

**Deliverable:** - Backup automatizzato - Test restore - Disaster recovery plan

---

#### 1.5.3.2 Task 4.3.2: High Availability

**Priorità:** MEDIA

**Obiettivo:** 99.9% uptime

**Approccio:** 1. Load balancer 2. Multiple application servers 3. Database replication 4. Failover automatico 5. Health checks

**Deliverable:** - HA setup - Failover testato - Monitoring HA

---

## 1.6 📅 FASE 5: FEATURES ENTERPRISE

### 1.6.1 5.1 MULTI-TENANCY (se necessario)

#### 1.6.1.1 Task 5.1.1: Tenant Isolation

**Priorità:** BASSA (se non necessario)

**Obiettivo:** Supporto multi-tenant sicuro

**Approccio:** 1. Database per tenant o schema isolation 2. Row-level security 3. Tenant context middleware 4. Testing isolation

---

### 1.6.2 5.2 API GATEWAY

#### 1.6.2.1 Task 5.2.1: API Gateway Enterprise

**Priorità:** MEDIA

**Obiettivo:** Gestione API centralizzata

**Approccio:** 1. Kong / AWS API Gateway 2. Rate limiting centralizzato 3. API versioning 4. Analytics API

---

### 1.6.3 5.3 ADVANCED FEATURES

#### 1.6.3.1 Task 5.3.1: Real-time Updates

**Priorità:** BASSA

**Obiettivo:** WebSocket per aggiornamenti real-time

---

#### 1.6.3.2 Task 5.3.2: Advanced Reporting

**Priorità:** BASSA

**Obiettivo:** Dashboard analytics avanzate

---

## 1.7 📊 METRICHE SUCCESSO

### 1.7.1 Sicurezza

- ✅ 0 vulnerabilità critiche/alte
- ✅ Security audit passato
- ✅ Penetration test passato
- ✅ Compliance GDPR/PCI-DSS

### 1.7.2 Performance

- ✅ Latenza API <100ms (p95)
- ✅ Throughput 3-5x migliorato
- ✅ Database CPU <50% sotto carico normale
- ✅ Cache hit rate >80%

### 1.7.3 Qualità

- ✅ Test coverage >80%
- ✅ Code quality score >8/10
- ✅ Zero bug critici in produzione
- ✅ Documentation coverage 100%

### 1.7.4 Affidabilità

- ✅ Uptime 99.9%
- ✅ MTTR <1 ora
- ✅ RTO <4 ore
- ✅ RPO <1 ora

---

## 1.8 ⚠️ RISCHI E MITIGAZIONE

### 1.8.1 Rischio 1: Downtime durante migrazione

**Mitigazione:** Blue-green deployment, migrazione graduale

### 1.8.2 Rischio 2: Regressioni funzionali

**Mitigazione:** Test suite completa, testing UAT estensivo

### 1.8.3 Rischio 3: Performance degradate

**Mitigazione:** Benchmark continui, monitoring performance

### 1.8.4 Rischio 4: Budget overrun

**Mitigazione:** Sprint planning accurato, review settimanali

---

## 1.9 📋 CHECKLIST IMPLEMENTAZIONE

### 1.9.1 Fase 1 - Sicurezza

- [ ] Prepared statements su tutte le query
- [ ] Credenziali in file sicuro
- [ ] Password hashing Argon2id
- [ ] CSRF protection completo
- [ ] XSS prevention completo
- [ ] File upload sicuro
- [ ] Session security
- [ ] Rate limiting

### 1.9.2 Fase 2 - Architettura

- [ ] Framework MVC implementato
- [ ] ORM/Query Builder
- [ ] API RESTful
- [ ] Database ottimizzato
- [ ] Caching implementato
- [ ] Connection pooling

### 1.9.3 Fase 3 - Qualità

- [ ] Unit tests >80% coverage
- [ ] Integration tests
- [ ] Security testing passato
- [ ] Code standards
- [ ] Documentazione completa

### 1.9.4 Fase 4 - Infrastruttura

- [ ] Monitoring completo
- [ ] Logging strutturato
- [ ] CI/CD pipeline
- [ ] Backup automatizzato
- [ ] High availability

---

**Documento creato:** 2025-11-24
**Versione:** 2.0
**Autore:** CTO DigitalBrain