PHARMDISPLAY - SECURITY DESIGN & DEPLOYMENT

SICUREZZA, INFRASTRUTTURA E DEPLOYMENT AUTOMATION

Versione: 1.0
Data: 05 Novembre 2025
Documento: 3 di 4

INDICE

1. SECURITY DESIGN

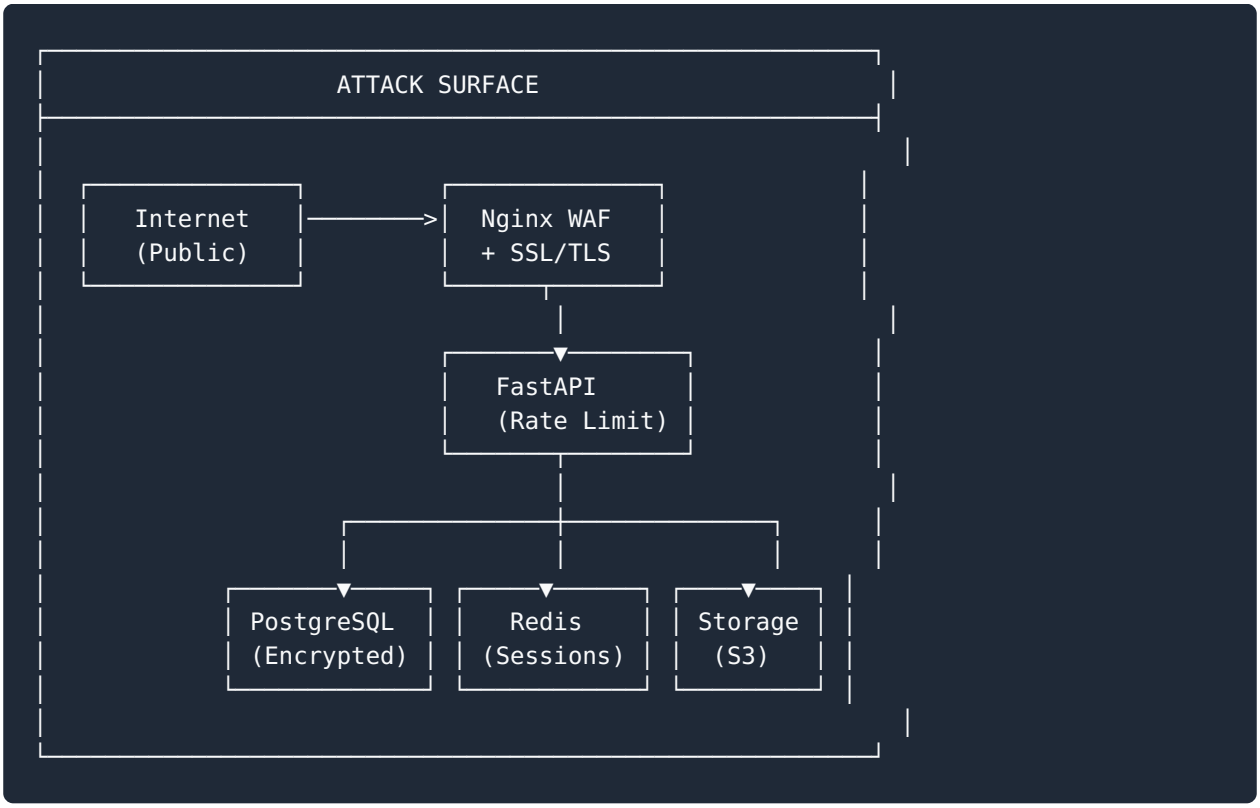
1.1 Threat Model

1.1.1 Asset Identification

Asset	Criticità	Minacce Principali
Database PostgreSQL	CRITICA	SQL Injection, Data breach, Ransomware
JWT Tokens	ALTA	Token theft, Session hijacking
User Passwords	ALTA	Brute force, Credential stuffing

Asset	Criticità	Minacce Principali
Device Activation Codes	MEDIA	Code prediction, Unauthorized activation
API Endpoints	MEDIA	DDoS, Rate limit bypass
Pharmacy Data	ALTA	Privacy violation (GDPR)
Admin Dashboard	CRITICA	Unauthorized access, Privilege escalation

1.1.2 Attack Vectors



1.2 Authentication & Authorization

1.2.1 JWT Implementation

```
# app/utils/security.py
from datetime import datetime, timedelta
from typing import Optional
from jose import JWTError, jwt
from passlib.context import CryptContext

# Configuration
SECRET_KEY = "your-secret-key-here-change-in-production"  # 64+ char random
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_HOURS = 24
```

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    """Generate JWT token"""
    to_encode = data.copy()

    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(hours=ACCESS_TOKEN_EXPIRE_HOURS)

    to_encode.update({
        "exp": expire,
        "iat": datetime.utcnow(),
        "jti": str(uuid.uuid4())  # JWT ID for revocation
    })

    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify password against hash"""
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    """Hash password with bcrypt (cost factor 12)"""
    return pwd_context.hash(password)

def decode_token(token: str) -> dict:
    """Decode and validate JWT token"""
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload
    except JWTError:
        raise HTTPException(
            status_code=401,
            detail="Could not validate credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
```

1.2.2 Password Policy

```
# app/utils/validators.py
import re
from pydantic import validator

class PasswordPolicy:
    """
    Password Requirements:
    - Minimum 8 characters
    - At least 1 uppercase letter
    - At least 1 lowercase letter
```

```
    - At least 1 number
    - At least 1 special character
    """

    MIN_LENGTH = 8
    PATTERN = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'

    @staticmethod
    def validate(password: str) -> bool:
        if len(password) < PasswordPolicy.MIN_LENGTH:
            raise ValueError(f"Password must be at least {PasswordPolicy.MIN_LENGTH} characters")

        if not re.match(PasswordPolicy.PATTERN, password):
            raise ValueError(
                "Password must contain: uppercase, lowercase, number, special character"
            )

        return True
```

1.2.3 Role-Based Access Control (RBAC)

```
# app/dependencies.py
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from sqlalchemy.orm import Session

security = HTTPBearer()

def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: Session = Depends(get_db)
) -> User:
    """Extract and validate current user from JWT"""
    token = credentials.credentials
    payload = decode_token(token)

    user_id = payload.get("sub")
    if user_id is None:
        raise HTTPException(status_code=401, detail="Invalid token")

    user = db.query(User).filter(User.id == user_id).first()
    if user is None or not user.is_active:
        raise HTTPException(status_code=401, detail="User not found")

    return user

def require_admin(current_user: User = Depends(get_current_user)) -> User:
    """Require admin role"""
    if current_user.role != UserRole.ADMIN:
        raise HTTPException(
            status_code=403,
            detail="Admin privileges required"
        )
```

```
    return current_user

def require_pharmacy_access(
    pharmacy_id: str,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
) -> Pharmacy:
    """Verify user has access to pharmacy"""
    pharmacy = db.query(Pharmacy).filter(Pharmacy.id == pharmacy_id).first()

    if not pharmacy:
        raise HTTPException(status_code=404, detail="Pharmacy not found")

    # Admin can access all pharmacies
    if current_user.role == UserRole.ADMIN:
        return pharmacy

    # User can only access own pharmacies
    if pharmacy.user_id != current_user.id:
        raise HTTPException(status_code=403, detail="Access denied")

    return pharmacy
```

1.3 Input Validation & Sanitization

1.3.1 Pydantic Schemas

```
# app/schemas/shift.py
from datetime import date, time
from typing import Optional
from pydantic import BaseModel, validator, Field

class ShiftCreate(BaseModel):
    date: date = Field(..., description="Shift date")
    start_time: time = Field(..., description="Start time (HH:MM)")
    end_time: time = Field(..., description="End time (HH:MM)")
    is_recurring: bool = Field(default=False)
    recurrence_rule: Optional[str] = Field(None, max_length=255)
    notes: Optional[str] = Field(None, max_length=500)

    @validator('end_time')
    def validate_time_range(cls, end_time, values):
        """Ensure end_time > start_time"""
        if 'start_time' in values and end_time <= values['start_time']:
            raise ValueError('end_time must be after start_time')
        return end_time

    @validator('recurrence_rule')
    def validate_recurrence(cls, rrule, values):
        """Validate RRULE format if recurring"""
        if values.get('is_recurring'):
            if not rrule:
```

```
                raise ValueError('recurrence_rule required when is_recurring=True')
            # Validate RRULE format (RFC 5545)
            try:
                from dateutil.rrule import rrulestr
                rrulestr(f"DTSTART:20240101\nRRULE:{rrule}")
            except Exception as e:
                raise ValueError(f"Invalid RRULE format: {e}")
        return rrule

    class Config:
        json_schema_extra = {
            "example": {
                "date": "2025-11-10",
                "start_time": "08:00",
                "end_time": "20:00",
                "is_recurring": True,
                "recurrence_rule": "FREQ=WEEKLY;BYDAY=MO,WE,FR",
                "notes": "Turno settimanale"
            }
        }
```

1.3.2 SQL Injection Prevention

```
# ALWAYS use SQLAlchemy ORM - NEVER raw SQL
from sqlalchemy.orm import Session

# ✅ SAFE - Parameterized query via ORM
def get_shifts_safe(db: Session, pharmacy_id: str, date_from: date):
    return db.query(Shift).filter(
        Shift.pharmacy_id == pharmacy_id,
        Shift.date >= date_from
    ).all()

# ❌ UNSAFE - Raw SQL with string concatenation (NEVER DO THIS)
def get_shifts_unsafe(db: Session, pharmacy_id: str):
    # VULNERABILITY: SQL Injection possible
    query = f"SELECT * FROM shifts WHERE pharmacy_id = '{pharmacy_id}'"
    return db.execute(query).fetchall()
```

1.4 API Rate Limiting

1.4.1 Redis-based Rate Limiter

```
# app/middleware/rate_limit.py
from fastapi import Request, HTTPException
from redis import Redis
import time

redis_client = Redis(host='localhost', port=6379, db=0, decode_responses=True)
```

```
class RateLimiter:
    """Token bucket rate limiter"""

    def __init__(self, requests_per_minute: int = 100):
        self.limit = requests_per_minute
        self.window = 60  # seconds

    async def check_rate_limit(self, request: Request, identifier: str):
        """Check if request is within rate limit"""
        key = f"rate_limit:{identifier}:{int(time.time() / self.window)}"

        current = redis_client.get(key)

        if current is None:
            # First request in this window
            redis_client.setex(key, self.window, 1)
            return True

        current = int(current)

        if current >= self.limit:
            raise HTTPException(
                status_code=429,
                detail=f"Rate limit exceeded. Max {self.limit} requests per minute.",
                headers={
                    "Retry-After": str(self.window),
                    "X-RateLimit-Limit": str(self.limit),
                    "X-RateLimit-Remaining": "0"
                }
            )

        # Increment counter
        redis_client.incr(key)
        return True

# Apply to routes
rate_limiter = RateLimiter(requests_per_minute=100)

@app.middleware("http")
async def rate_limit_middleware(request: Request, call_next):
    """Apply rate limiting to all requests"""

    # Get identifier (user ID or IP address)
    if hasattr(request.state, "user"):
        identifier = str(request.state.user.id)
    else:
        identifier = request.client.host

    # Public display endpoints have higher limit
    if request.url.path.startswith("/api/v1/display"):
        limiter = RateLimiter(requests_per_minute=1000)
    else:
        limiter = rate_limiter

    await limiter.check_rate_limit(request, identifier)
```

```
    response = await call_next(request)
    return response
```

1.5 HTTPS & SSL/TLS

1.5.1 Nginx Configuration with Let's Encrypt

```
# /etc/nginx/sites-available/pharmdisplay

# HTTP -> HTTPS redirect
server {
    listen 80;
    listen [::]:80;
    server_name api.pharmdisplay.com;

    # Let's Encrypt ACME challenge
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    # Redirect all HTTP to HTTPS
    location / {
        return 301 https://$server_name$request_uri;
    }
}

# HTTPS server
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name api.pharmdisplay.com;

    # SSL certificates
    ssl_certificate /etc/letsencrypt/live/api.pharmdisplay.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.pharmdisplay.com/privkey.pem;

    # SSL configuration (Mozilla Intermediate)
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GC
    ssl_prefer_server_ciphers off;

    # OCSP stapling
    ssl_stapling on;
    ssl_stapling_verify on;
    ssl_trusted_certificate /etc/letsencrypt/live/api.pharmdisplay.com/chain.pem;

    # Security headers
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
```

```
    add_header Referrer-Policy "strict-origin-when-cross-origin" always;

    # CORS headers (adjust for production)
    add_header Access-Control-Allow-Origin "https://dashboard.pharmdisplay.com" always;
    add_header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS" always;
    add_header Access-Control-Allow-Headers "Authorization, Content-Type" always;
    add_header Access-Control-Max-Age 3600 always;

    # Proxy to FastAPI
    location /api/v1/ {
        proxy_pass http://127.0.0.1:8000;
        proxy_http_version 1.1;

        # Headers
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # WebSocket support
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api_limit:10m rate=100r/m;
    limit_req zone=api_limit burst=20 nodelay;
}
```

1.5.2 Let's Encrypt Auto-renewal

```
#!/bin/bash
# /usr/local/bin/renew-ssl.sh

# Renew certificates
certbot renew --nginx --quiet

# Reload nginx if renewed
if [ $? -eq 0 ]; then
    nginx -s reload
fi
```

```
# Cron job: daily at 3am
0 3 * * * /usr/local/bin/renew-ssl.sh >> /var/log/letsencrypt-renew.log 2>&1
```

1.6 Data Encryption

1.6.1 Database Encryption at Rest

```
# PostgreSQL disk encryption with LUKS
sudo cryptsetup luksFormat /dev/sdb
sudo cryptsetup luksOpen /dev/sdb pgdata
sudo mkfs.ext4 /dev/mapper/pgdata
sudo mount /dev/mapper/pgdata /var/lib/postgresql
```

1.6.2 Environment Variables Security

```
# app/config.py
from pydantic_settings import BaseSettings
from typing import Optional

class Settings(BaseSettings):
    """Application settings from environment variables"""

    # Database
    DATABASE_URL: str

    # Redis
    REDIS_URL: str = "redis://localhost:6379/0"

    # Security
    SECRET_KEY: str  # Must be 64+ chars random
    ALGORITHM: str = "HS256"
    ACCESS_TOKEN_EXPIRE_HOURS: int = 24

    # Email
    SMTP_HOST: Optional[str] = None
    SMTP_PORT: int = 587
    SMTP_USERNAME: Optional[str] = None
    SMTP_PASSWORD: Optional[str] = None
    ADMIN_EMAIL: str = "admin@pharmdisplay.com"

    # Storage
    S3_BUCKET: Optional[str] = None
    S3_ACCESS_KEY: Optional[str] = None
    S3_SECRET_KEY: Optional[str] = None

    # Sentry
    SENTRY_DSN: Optional[str] = None

    class Config:
        env_file = ".env"
        case_sensitive = True

settings = Settings()
```

```
# .env.example (NEVER commit .env to git)
DATABASE_URL=postgresql://pharmdisplay:CHANGE_ME@localhost:5432/pharmdisplay
REDIS_URL=redis://localhost:6379/0
SECRET_KEY=GENERATE_64_CHAR_RANDOM_STRING_HERE
SMTP_HOST=smtp.gmail.com
SMTP_USERNAME=noreply@pharmdisplay.com
SMTP_PASSWORD=CHANGE_ME
ADMIN_EMAIL=admin@pharmdisplay.com
```

---

2. INFRASTRUCTURE AS CODE

2.1 VPS Provider Recommendation

Comparison Matrix

Provider	CPU	RAM	Storage	Bandwidth	Price/Month	Setup Time
Hetzner CX21	2 vCPU	4GB	40GB SSD	20TB	**€5.83**	1min
Contabo VPS S	4 vCPU	8GB	200GB SSD	32TB	€6.99	24h
OVH VPS Starter	1 vCPU	2GB	20GB SSD	Unlimited	€6.00	2min
Register.it VPS	2 vCPU	4GB	80GB SSD	1TB	€49.00	24h

🏆 RACCOMANDAZIONE: Hetzner CX21

Motivazioni:

- ✅ **Miglior rapporto qualità/prezzo**: €5.83/mese
- ✅ **Setup istantaneo**: server pronto in 1 minuto
- ✅ **Datacenter europei**: conformità GDPR garantita
- ✅ **Affidabilità provata**: uptime 99.9%+
- ✅ **Bandwidth generosa**: 20TB/mese sufficienti per 100+ devices
- ✅ **Snapshot gratuiti**: backup integrati
- ✅ **IPv4 + IPv6**: supporto completo
- ✅ **API completa**: automazione deployment

Alternative per scalabilità futura:

- **< 20 farmacie**: Hetzner CX21 (€5.83/mese)

- **20-50 farmacie**: Hetzner CX31 (€11.90/mese - 2 vCPU, 8GB RAM)
- **50-100 farmacie**: Hetzner CX41 (€24.50/mese - 4 vCPU, 16GB RAM)

2.2 Server Setup Script (Ubuntu 24.04)

```
#!/bin/bash
# setup-server.sh - Automated VPS setup for PharmDisplay

set -e  # Exit on error

echo "======================================"
echo "PharmDisplay Server Setup"
echo "======================================"

# Update system
apt-get update
apt-get upgrade -y

# Install dependencies
apt-get install -y \
    nginx \
    postgresql-15 \
    postgresql-contrib \
    postgis \
    redis-server \
    python3.11 \
    python3.11-venv \
    python3-pip \
    git \
    ufw \
    certbot \
    python3-certbot-nginx \
    supervisor \
    tesseract-ocr \
    tesseract-ocr-ita

# Configure firewall
ufw default deny incoming
ufw default allow outgoing
ufw allow ssh
ufw allow http
ufw allow https
ufw --force enable

# Create pharmdisplay user
useradd -m -s /bin/bash pharmdisplay
mkdir -p /home/pharmdisplay/app
chown pharmdisplay:pharmdisplay /home/pharmdisplay/app

# Setup PostgreSQL
sudo -u postgres psql <<EOF
CREATE DATABASE pharmdisplay;
CREATE USER pharmdisplay WITH PASSWORD 'CHANGE_ME_IN_PRODUCTION';
```

```
GRANT ALL PRIVILEGES ON DATABASE pharmdisplay TO pharmdisplay;
\c pharmdisplay
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "postgis";
CREATE EXTENSION IF NOT EXISTS "pg_trgm";
EOF

# Configure PostgreSQL for network connections
cat >> /etc/postgresql/15/main/pg_hba.conf <<EOF
host    pharmdisplay    pharmdisplay    127.0.0.1/32    scram-sha-256
EOF

systemctl restart postgresql

# Configure Redis
sed -i 's/^bind 127.0.0.1/bind 127.0.0.1/' /etc/redis/redis.conf
sed -i 's/# maxmemory <bytes>/maxmemory 512mb/' /etc/redis/redis.conf
sed -i 's/# maxmemory-policy noeviction/maxmemory-policy allkeys-lru/' /etc/redis/redis.conf
systemctl restart redis-server

# Setup Python virtual environment
sudo -u pharmdisplay python3.11 -m venv /home/pharmdisplay/venv
sudo -u pharmdisplay /home/pharmdisplay/venv/bin/pip install --upgrade pip

echo "======================================"
echo "Server setup complete!"
echo "======================================"
echo "Next steps:"
echo "1. Deploy application code"
echo "2. Configure SSL with certbot"
echo "3. Setup systemd services"
echo "======================================"
```

2.3 Application Deployment

```
#!/bin/bash
# deploy.sh - Application deployment script

DEPLOY_USER="pharmdisplay"
APP_DIR="/home/$DEPLOY_USER/app"
VENV_DIR="/home/$DEPLOY_USER/venv"
GIT_REPO="https://github.com/yourusername/pharmdisplay-backend.git"
BRANCH="main"

echo "Deploying PharmDisplay..."

# Pull latest code
cd $APP_DIR
sudo -u $DEPLOY_USER git pull origin $BRANCH

# Install dependencies
sudo -u $DEPLOY_USER $VENV_DIR/bin/pip install -r requirements.txt
```

```
# Run database migrations
sudo -u $DEPLOY_USER $VENV_DIR/bin/alembic upgrade head

# Restart services
systemctl restart pharmdisplay-api
systemctl restart pharmdisplay-celery

echo "Deployment complete!"
```

2.4 Systemd Services

```
# /etc/systemd/system/pharmdisplay-api.service
[Unit]
Description=PharmDisplay FastAPI Application
After=network.target postgresql.service redis.service

[Service]
Type=notify
User=pharmdisplay
Group=pharmdisplay
WorkingDirectory=/home/pharmdisplay/app
Environment="PATH=/home/pharmdisplay/venv/bin"
ExecStart=/home/pharmdisplay/venv/bin/gunicorn \
    -k uvicorn.workers.UvicornWorker \
    -w 4 \
    -b 127.0.0.1:8000 \
    --access-logfile /var/log/pharmdisplay/access.log \
    --error-logfile /var/log/pharmdisplay/error.log \
    app.main:app

Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
# /etc/systemd/system/pharmdisplay-celery.service
[Unit]
Description=PharmDisplay Celery Worker
After=network.target redis.service

[Service]
Type=forking
User=pharmdisplay
Group=pharmdisplay
WorkingDirectory=/home/pharmdisplay/app
Environment="PATH=/home/pharmdisplay/venv/bin"
ExecStart=/home/pharmdisplay/venv/bin/celery -A app.tasks worker \
    --loglevel=info \
    --logfile=/var/log/pharmdisplay/celery.log \
    --concurrency=2
```

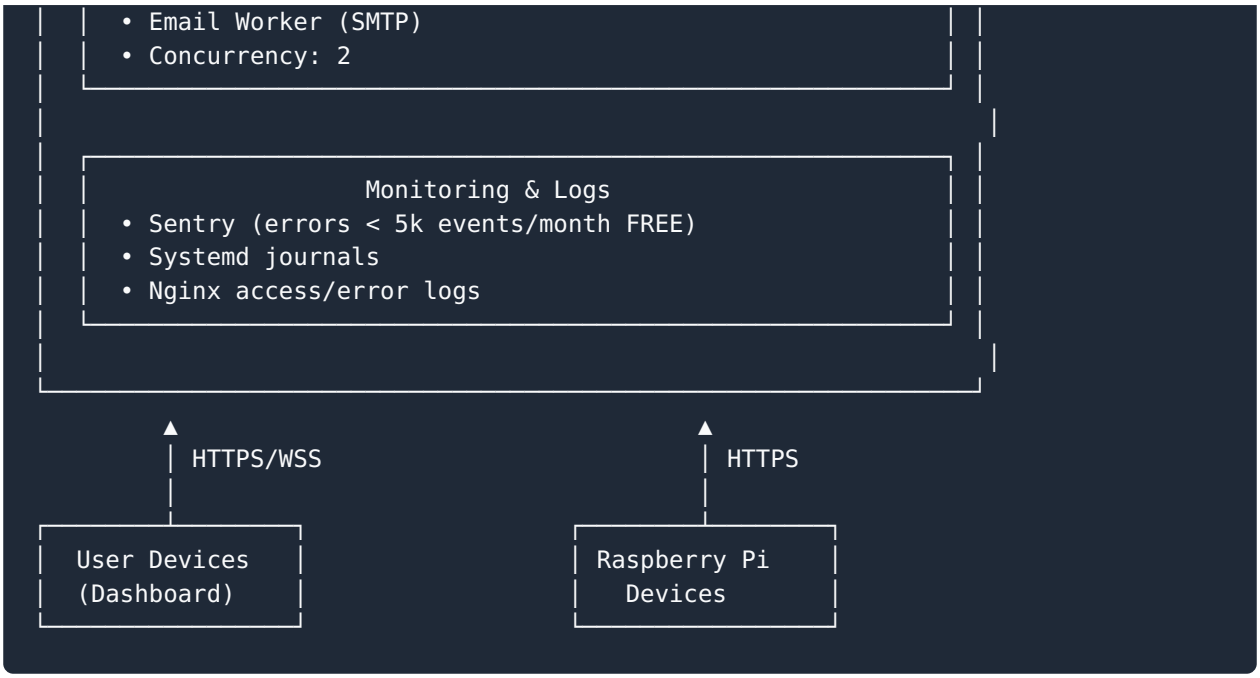
```
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

---

3. DEPLOYMENT ARCHITECTURE

3.1 Production Architecture Diagram



3.2 CI/CD Pipeline (GitHub Actions)

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [ main ]
  workflow_dispatch:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          pip install -r requirements.txt
          pip install pytest pytest-cov

      - name: Run tests
        run: pytest --cov=app tests/

      - name: Check code quality
        run: |
          pip install black flake8
          black --check app/
```

```
          flake8 app/ --max-line-length=100

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - name: Deploy to server
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /home/pharmdisplay/app
            git pull origin main
            source /home/pharmdisplay/venv/bin/activate
            pip install -r requirements.txt
            alembic upgrade head
            sudo systemctl restart pharmdisplay-api
            sudo systemctl restart pharmdisplay-celery
```

3.3 Database Backup Strategy

```
#!/bin/bash
# /usr/local/bin/backup-database.sh

BACKUP_DIR="/var/backups/pharmdisplay"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="pharmdisplay_$DATE.sql.gz"
S3_BUCKET="s3://pharmdisplay-backups"

# Create backup directory
mkdir -p $BACKUP_DIR

# Dump database
pg_dump -U pharmdisplay pharmdisplay | gzip > "$BACKUP_DIR/$BACKUP_FILE"

# Upload to S3 (if configured)
if command -v aws &> /dev/null; then
    aws s3 cp "$BACKUP_DIR/$BACKUP_FILE" "$S3_BUCKET/"
fi

# Keep only last 7 days locally
find $BACKUP_DIR -name "*.sql.gz" -mtime +7 -delete

echo "Backup completed: $BACKUP_FILE"
```

```
# Cron job: daily at 2am
0 2 * * * /usr/local/bin/backup-database.sh >> /var/log/pharmdisplay-backup.log 2>&1
```

---

4. MONITORING & ALERTING

4.1 Sentry Integration

```
# app/main.py
import sentry_sdk
from sentry_sdk.integrations.fastapi import FastApiIntegration
from app.config import settings

if settings.SENTRY_DSN:
    sentry_sdk.init(
        dsn=settings.SENTRY_DSN,
        integrations=[FastApiIntegration()],
        traces_sample_rate=0.1,  # 10% of requests
        environment="production",
        release="pharmdisplay@1.0.0"
    )

app = FastAPI()
```

4.2 Health Check Endpoints

```
# app/api/v1/health.py
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from redis import Redis
from app.database import get_db
from app.config import settings

router = APIRouter()

@router.get("/health")
async def health_check():
    """Basic health check"""
    return {"status": "healthy"}

@router.get("/health/detailed")
async def detailed_health_check(db: Session = Depends(get_db)):
    """Detailed health check with dependencies"""
    status = {
        "api": "healthy",
        "database": "unknown",
        "redis": "unknown"
```

```
    }

    # Check database
    try:
        db.execute("SELECT 1")
        status["database"] = "healthy"
    except Exception as e:
        status["database"] = f"unhealthy: {str(e)}"

    # Check Redis
    try:
        redis_client = Redis.from_url(settings.REDIS_URL)
        redis_client.ping()
        status["redis"] = "healthy"
    except Exception as e:
        status["redis"] = f"unhealthy: {str(e)}"

    return status
```

4.3 Email Notification System

```
# app/utils/email.py
from fastapi_mail import FastMail, MessageSchema, ConnectionConfig
from app.config import settings

conf = ConnectionConfig(
    MAIL_USERNAME=settings.SMTP_USERNAME,
    MAIL_PASSWORD=settings.SMTP_PASSWORD,
    MAIL_FROM=settings.SMTP_USERNAME,
    MAIL_PORT=settings.SMTP_PORT,
    MAIL_SERVER=settings.SMTP_HOST,
    MAIL_STARTTLS=True,
    MAIL_SSL_TLS=False,
    USE_CREDENTIALS=True
)

fm = FastMail(conf)

async def send_device_offline_alert(device_id: str, pharmacy_name: str):
    """Send email when device goes offline"""

    message = MessageSchema(
        subject=f"⚠️ Device Offline: {pharmacy_name}",
        recipients=[settings.ADMIN_EMAIL],  # Admin email only
        body=f"""
        <h2>Device Offline Alert</h2>
        <p>Device <strong>{device_id}</strong> at <strong>{pharmacy_name}</strong> has gone offl
        <p>Last seen: {datetime.utcnow()}</p>
        <p><a href="https://dashboard.pharmdisplay.com/devices/{device_id}">View Device</a></p>
        """,
        subtype="html"
    )
```

```
    await fm.send_message(message)

async def send_user_notification(user_email: str, subject: str, body: str):
    """Send notification to user (if email configured)"""

    if not user_email:
        return  # User has no email configured

    message = MessageSchema(
        subject=subject,
        recipients=[user_email],
        body=body,
        subtype="html"
    )

    await fm.send_message(message)
```

4.4 Dashboard Alert System

```
# app/api/v1/notifications.py
from fastapi import APIRouter, Depends, WebSocket
from sqlalchemy.orm import Session
from app.dependencies import get_current_user

router = APIRouter()

# In-memory notification queue (use Redis in production)
notification_queue = {}

@router.websocket("/ws/notifications")
async def websocket_notifications(
    websocket: WebSocket,
    current_user: User = Depends(get_current_user)
):
    """WebSocket for real-time notifications"""
    await websocket.accept()

    user_id = str(current_user.id)

    try:
        while True:
            # Check for new notifications
            if user_id in notification_queue:
                notifications = notification_queue[user_id]
                await websocket.send_json({"notifications": notifications})
                notification_queue[user_id] = []

            await asyncio.sleep(5)  # Poll every 5 seconds

    except WebSocketDisconnect:
        pass

async def push_notification(user_id: str, notification: dict):
```

```
    """Push notification to user's queue"""
    if user_id not in notification_queue:
        notification_queue[user_id] = []

    notification_queue[user_id].append({
        **notification,
        "timestamp": datetime.utcnow().isoformat()
    })
```

---

5. DISASTER RECOVERY

5.1 Recovery Time Objective (RTO)

Scenario	RTO Target	Recovery Steps
Database corruption	< 4 hours	Restore from daily backup
Complete server failure	< 8 hours	Provision new VPS, restore from backup
Security breach	< 1 hour	Rotate all secrets, force re-authentication
DDoS attack	< 30 min	Enable Cloudflare DDoS protection

5.2 Backup Verification

```
#!/bin/bash
# /usr/local/bin/verify-backup.sh

BACKUP_DIR="/var/backups/pharmdisplay"
LATEST_BACKUP=$(ls -t $BACKUP_DIR/*.sql.gz | head -1)

echo "Verifying backup: $LATEST_BACKUP"

# Test database restore to temporary database
createdb -U postgres pharmdisplay_test
gunzip -c $LATEST_BACKUP | psql -U postgres pharmdisplay_test > /dev/null 2>&1

if [ $? -eq 0 ]; then
    echo "✓ Backup verification successful"
    dropdb -U postgres pharmdisplay_test
    exit 0
else
    echo "✗ Backup verification FAILED"
    dropdb -U postgres pharmdisplay_test
    # Send alert email
```

```
    exit 1
fi
```

5.3 Disaster Recovery Runbook

```
# PharmDisplay Disaster Recovery Runbook

## Scenario 1: Database Corruption

1. Stop all services:
   ```bash
   systemctl stop pharmdisplay-api pharmdisplay-celery
```

1. Backup corrupted database:

   ```
   pg_dump pharmdisplay > /tmp/corrupted_$(date +%s).sql
   ```

2. Drop and recreate database:

   ```
   dropdb pharmdisplay
   createdb pharmdisplay
   ```

3. Restore from latest backup:

   ```
   gunzip -c /var/backups/pharmdisplay/latest.sql.gz | psql pharmdisplay
   ```

4. Restart services:

   ```
   systemctl start pharmdisplay-api pharmdisplay-celery
   ```

5. Verify system health:

   ```
   curl https://api.pharmdisplay.com/api/v1/health/detailed
   ```

Scenario 2: Complete Server Failure

1. Provision new Hetzner VPS (CX21)

2. Run setup script:
   ```

```
bash setup-server.sh
```

3. Configure DNS to point to new IP
4. Deploy application:

```
git clone https://github.com/yourusername/pharmdisplay-backend.git
cd pharmdisplay-backend
bash deploy.sh
```

5. Restore database from S3:

```
aws s3 cp s3://pharmdisplay-backups/latest.sql.gz /tmp/
gunzip -c /tmp/latest.sql.gz | psql pharmdisplay
```

6. Configure SSL:

```
certbot --nginx -d api.pharmdisplay.com
```

7. Start services and verify

Estimated Total Time: 6-8 hours
```

---

# CHECKLIST PRE-PRODUZIONE

- [ ] Server Hetzner CX21 configurato
- [ ] Database PostgreSQL con PostGIS installato
- [ ] Redis configurato
- [ ] Nginx con SSL Let's Encrypt attivo
- [ ] Firewall UFW configurato
- [ ] Systemd services attivi e abilitati
- [ ] Backup automatici schedulati (cron)
- [ ] Sentry configurato per error tracking
- [ ] Email SMTP configurato
- [ ] DNS configurato (api.pharmdisplay.com)
- [ ] Secrets in variabili d'ambiente (no hardcoding)
- [ ] Rate limiting attivo
```

- [ ] CORS policies configurate
- [ ] Health check endpoints testati
- [ ] Disaster recovery runbook pronto
- [ ] Documentazione deployment aggiornata

Fine Documento 03 - Security & Deployment