Manual del Programador

Sistema de Auditoría de Seguridad Web

Versión 1.0

Fecha: Julio 2025

Tabla de Contenidos

Introducción Técnica

El Sistema de Auditoría de Seguridad Web es una aplicación full-stack desarrollada para evaluar y mejorar la calidad del código y la seguridad de aplicaciones web. Implementa un modelo de arquitectura MVC (Modelo-Vista-Controlador) con patrones de diseño modernos.

Objetivos del Sistema

- Análisis automatizado de código fuente
- Detección de vulnerabilidades de seguridad
- Generación de reportes técnicos detallados
- Interfaz web intuitiva para gestión de auditorías
- API REST para integración con otras herramientas

Principios de Desarrollo

- **Modularidad**: Componentes independientes y reutilizables
- **Escalabilidad**: Arquitectura que soporta crecimiento
- **Seguridad**: Implementación de mejores prácticas
- **Mantenibilidad**: Código limpio y bien documentado

Arquitectura del Sistema

Patrón MVC Implementado

Modelo (Model)

python

```python
# app/models/vulnerability.py
class Vulnerability:
    def __init__(self, id, name, severity, description, file_path, line_number):
        self.id = id
        self.name = name
        self.severity = severity
        self.description = description
        self.file_path = file_path
        self.line_number = line_number
```

Vista (View)

javascript

```javascript
// Frontend React Components
// components/Dashboard.jsx
import React from 'react';
import VulnerabilityList from './VulnerabilityList';
import StatisticsPanel from './StatisticsPanel';

const Dashboard = () => {
    return (
        <div className="dashboard">
            <StatisticsPanel />
            <VulnerabilityList />
        </div>
    );
};
```

Controlador (Controller)

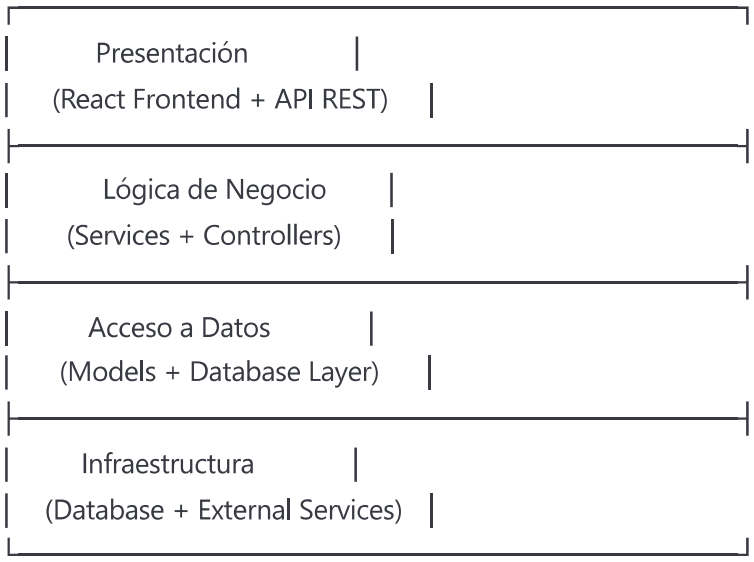
python

```python
# app/controllers/audit_controller.py
from flask import Blueprint, request, jsonify
from app.services.audit_service import AuditService

audit_bp = Blueprint('audit', __name__)

@audit_bp.route('/audit', methods=['POST'])
def create_audit():
    data = request.get_json()
    result = AuditService.create_audit(data)
    return jsonify(result)
```

Arquitectura de Capas



Tecnologías Utilizadas

Backend

- **Framework**: Flask 2.3.x (Python)
- **ORM**: SQLAlchemy 1.4.x
- **Base de Datos**: PostgreSQL 14.x
- **Autenticación**: Flask-JWT-Extended
- **Validación**: Marshmallow 3.x
- **Testing**: PyTest 7.x

Frontend

- **Framework**: React 18.x
- **Estado**: Redux Toolkit

- **Routing**: React Router 6.x
- **UI Components**: Material-UI 5.x
- **HTTP Client**: Axios
- **Testing**: Jest + React Testing Library

Herramientas de Análisis

- **Análisis Estático**: Bandit, SonarQube
- **Escaneo de Dependencias**: Safety, OWASP Dependency Check
- **Análisis de Código**: ESLint, PyLint
- **Pruebas de Seguridad**: OWASP ZAP, Nmap

DevOps

- **Contenedores**: Docker + Docker Compose
- **CI/CD**: GitHub Actions
- **Monitoring**: Prometheus + Grafana
- **Logs**: ELK Stack (Elasticsearch, Logstash, Kibana)

Estructura del Proyecto

```
sistema-auditoria-web/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── models/
│   │   │   ├── __init__.py
│   │   │   ├── user.py
│   │   │   ├── project.py
│   │   │   ├── audit.py
│   │   │   └── vulnerability.py
│   │   ├── controllers/
│   │   │   ├── __init__.py
│   │   │   ├── auth_controller.py
│   │   │   ├── project_controller.py
│   │   │   ├── audit_controller.py
│   │   │   └── report_controller.py
│   │   ├── services/
│   │   │   ├── __init__.py
│   │   │   ├── auth_service.py
│   │   │   ├── audit_service.py
│   │   │   ├── scanner_service.py
│   │   │   └── report_service.py
│   │   ├── utils/
│   │   │   ├── __init__.py
│   │   │   ├── database.py
│   │   │   ├── validators.py
│   │   │   └── helpers.py
│   │   └── static/
│   ├── migrations/
│   ├── tests/
│   ├── config.py
│   ├── requirements.txt
│   └── run.py
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   │   ├── common/
│   │   │   ├── dashboard/
│   │   │   ├── projects/
│   │   │   ├── audits/
│   │   │   └── reports/
│   │   ├── services/
│   │   ├── store/
│   │   ├── utils/
│   │   ├── App.js
│   │   └── index.js
```

```
│   ├── public/
│   ├── package.json
│   └── Dockerfile
├── docker-compose.yml
├── README.md
└── .env.example
```

Instalación y Configuración

Configuración del Entorno de Desarrollo

1. Clonar el Repositorio

```bash
git clone https://github.com/usuario/sistema-auditoria-web.git
cd sistema-auditoria-web
```

2. Configurar Variables de Entorno

```bash
# .env
DATABASE_URL=postgresql://usuario:password@localhost:5432/auditoria_db
JWT_SECRET_KEY=tu-clave-secreta-jwt
FLASK_ENV=development
FLASK_APP=run.py
REDIS_URL=redis://localhost:6379
```

3. Configurar Base de Datos

```bash
# Crear base de datos
createdb auditoria_db

# Ejecutar migraciones
cd backend
flask db upgrade
```

4. Instalar Dependencias Backend

```bash
```

```
cd backend
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
pip install -r requirements.txt
```

5. Instalar Dependencias Frontend

bash

```
cd frontend
npm install
```

6. Configurar Docker (Opcional)

bash

```
# Construir y ejecutar con Docker Compose
docker-compose up --build
```

Módulos del Sistema

Módulo de Autenticación

Modelo de Usuario

python

```python
# app/models/user.py
from werkzeug.security import generate_password_hash, check_password_hash
from app.utils.database import db

class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(20), default='user')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```

Servicio de Autenticación

```python
```

```python
# app/services/auth_service.py
from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity
from app.models.user import User

class AuthService:
    @staticmethod
    def login(username, password):
        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            access_token = create_access_token(identity=user.id)
            return {
                'access_token': access_token,
                'user': {
                    'id': user.id,
                    'username': user.username,
                    'email': user.email,
                    'role': user.role
                }
            }
        return None

    @staticmethod
    def register(username, email, password):
        if User.query.filter_by(username=username).first():
            return {'error': 'Username already exists'}

        user = User(username=username, email=email)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()

        return {'message': 'User created successfully'}
```

Módulo de Análisis de Código

Servicio de Escáner

python

```python
# app/services/scanner_service.py
import subprocess
import json
import os
from app.models.vulnerability import Vulnerability

class ScannerService:
    @staticmethod
    def scan_code(project_path, scan_type='all'):
        vulnerabilities = []

        if scan_type in ['all', 'static']:
            vulnerabilities.extend(ScannerService._bandit_scan(project_path))
            vulnerabilities.extend(ScannerService._safety_scan(project_path))

        if scan_type in ['all', 'dynamic']:
            vulnerabilities.extend(ScannerService._zap_scan(project_path))

        return vulnerabilities

    @staticmethod
    def _bandit_scan(project_path):
        """Análisis estático con Bandit"""
        try:
            result = subprocess.run([
                'bandit', '-r', project_path, '-f', 'json'
            ], capture_output=True, text=True)

            if result.returncode == 0:
                data = json.loads(result.stdout)
                vulnerabilities = []

                for item in data.get('results', []):
                    vuln = Vulnerability(
                        name=item['test_id'],
                        severity=item['issue_severity'],
                        description=item['issue_text'],
                        file_path=item['filename'],
                        line_number=item['line_number']
                    )
                    vulnerabilities.append(vuln)

                return vulnerabilities
        except Exception as e:
            print(f"Error en Bandit scan: {e}")
```

```
        return []

    @staticmethod
    def _safety_scan(project_path):
        """Análisis de dependencias con Safety"""
        try:
            result = subprocess.run([
                'safety', 'check', '--json'
            ], capture_output=True, text=True, cwd=project_path)

            if result.returncode != 0:
                data = json.loads(result.stdout)
                vulnerabilities = []

                for item in data:
                    vuln = Vulnerability(
                        name=f"Dependency: {item['package']}",
                        severity='HIGH',
                        description=item['advisory'],
                        file_path='requirements.txt',
                        line_number=0
                    )
                    vulnerabilities.append(vuln)

                return vulnerabilities
        except Exception as e:
            print(f"Error en Safety scan: {e}")

        return []
```

Módulo de Generación de Reportes

Servicio de Reportes

python

```python
# app/services/report_service.py
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table
from reportlab.lib.styles import getSampleStyleSheet
from jinja2 import Environment, FileSystemLoader

class ReportService:
    @staticmethod
    def generate_pdf_report(audit_id, vulnerabilities):
        """Genera reporte en PDF"""
        filename = f"audit_report_{audit_id}.pdf"
        doc = SimpleDocTemplate(filename, pagesize=letter)

        styles = getSampleStyleSheet()
        story = []

        # Título
        title = Paragraph("Reporte de Auditoría de Seguridad", styles['Title'])
        story.append(title)
        story.append(Spacer(1, 20))

        # Resumen
        summary_data = ReportService._generate_summary(vulnerabilities)
        summary_table = Table(summary_data)
        story.append(summary_table)
        story.append(Spacer(1, 20))

        # Detalles de vulnerabilidades
        for vuln in vulnerabilities:
            vuln_title = Paragraph(f"Vulnerabilidad: {vuln.name}", styles['Heading2'])
            story.append(vuln_title)

            vuln_details = Paragraph(f"""
                <b>Severidad:</b> {vuln.severity}<br/>
                <b>Archivo:</b> {vuln.file_path}<br/>
                <b>Línea:</b> {vuln.line_number}<br/>
                <b>Descripción:</b> {vuln.description}
            """, styles['Normal'])
            story.append(vuln_details)
            story.append(Spacer(1, 12))

        doc.build(story)
        return filename

    @staticmethod
    def generate_html_report(audit_id, vulnerabilities):
```

```python
        """Genera reporte en HTML"""
        env = Environment(loader=FileSystemLoader('templates'))
        template = env.get_template('report_template.html')

        context = {
            'audit_id': audit_id,
            'vulnerabilities': vulnerabilities,
            'summary': ReportService._generate_summary(vulnerabilities)
        }

        html_content = template.render(context)
        filename = f"audit_report_{audit_id}.html"

        with open(filename, 'w', encoding='utf-8') as f:
            f.write(html_content)

        return filename

    @staticmethod
    def _generate_summary(vulnerabilities):
        """Genera resumen de vulnerabilidades"""
        summary = {
            'total': len(vulnerabilities),
            'critical': len([v for v in vulnerabilities if v.severity == 'CRITICAL']),
            'high': len([v for v in vulnerabilities if v.severity == 'HIGH']),
            'medium': len([v for v in vulnerabilities if v.severity == 'MEDIUM']),
            'low': len([v for v in vulnerabilities if v.severity == 'LOW'])
        }
        return summary
```

API y Endpoints

Autenticación

POST /api/auth/login

python

```python
@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()

    # Validación
    if not data or not data.get('username') or not data.get('password'):
        return jsonify({'error': 'Username and password required'}), 400

    # Autenticación
    result = AuthService.login(data['username'], data['password'])
    if result:
        return jsonify(result), 200
    else:
        return jsonify({'error': 'Invalid credentials'}), 401
```

POST /api/auth/register

python

```python
@auth_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()

    # Validación
    required_fields = ['username', 'email', 'password']
    if not all(field in data for field in required_fields):
        return jsonify({'error': 'All fields required'}), 400

    # Registro
    result = AuthService.register(data['username'], data['email'], data['password'])
    return jsonify(result), 201
```

Proyectos

GET /api/projects

python

```python
@project_bp.route('/', methods=['GET'])
@jwt_required()
def get_projects():
    user_id = get_jwt_identity()
    projects = Project.query.filter_by(user_id=user_id).all()

    return jsonify([{
        'id': p.id,
        'name': p.name,
        'description': p.description,
        'created_at': p.created_at.isoformat()
    } for p in projects])
```

POST /api/projects

python

```python
@project_bp.route('/', methods=['POST'])
@jwt_required()
def create_project():
    user_id = get_jwt_identity()
    data = request.get_json()

    project = Project(
        name=data['name'],
        description=data.get('description', ''),
        user_id=user_id
    )

    db.session.add(project)
    db.session.commit()

    return jsonify({
        'id': project.id,
        'name': project.name,
        'description': project.description
    }), 201
```

Auditorías

POST /api/audits

python

```python
@audit_bp.route('/', methods=['POST'])
@jwt_required()
def create_audit():
    data = request.get_json()

    # Validación
    if not data or not data.get('project_id'):
        return jsonify({'error': 'Project ID required'}), 400

    # Crear auditoría
    audit = Audit(
        project_id=data['project_id'],
        scan_type=data.get('scan_type', 'all'),
        status='pending'
    )

    db.session.add(audit)
    db.session.commit()

    # Ejecutar análisis asíncrono
    from app.tasks import run_security_scan
    run_security_scan.delay(audit.id)

    return jsonify({
        'id': audit.id,
        'status': audit.status,
        'created_at': audit.created_at.isoformat()
    }), 201
```

GET /api/audits/{audit_id}

python

```python
@audit_bp.route('/<int:audit_id>', methods=['GET'])
@jwt_required()
def get_audit(audit_id):
    audit = Audit.query.get_or_404(audit_id)

    # Verificar permisos
    if audit.project.user_id != get_jwt_identity():
        return jsonify({'error': 'Unauthorized'}), 403

    vulnerabilities = Vulnerability.query.filter_by(audit_id=audit_id).all()

    return jsonify({
        'id': audit.id,
        'project_id': audit.project_id,
        'status': audit.status,
        'scan_type': audit.scan_type,
        'created_at': audit.created_at.isoformat(),
        'vulnerabilities': [{
            'id': v.id,
            'name': v.name,
            'severity': v.severity,
            'description': v.description,
            'file_path': v.file_path,
            'line_number': v.line_number
        } for v in vulnerabilities]
    })
```

Reportes

GET /api/reports/{audit_id}/pdf

python

```python
@report_bp.route('/<int:audit_id>/pdf', methods=['GET'])
@jwt_required()
def generate_pdf_report(audit_id):
    audit = Audit.query.get_or_404(audit_id)

    # Verificar permisos
    if audit.project.user_id != get_jwt_identity():
        return jsonify({'error': 'Unauthorized'}), 403

    vulnerabilities = Vulnerability.query.filter_by(audit_id=audit_id).all()
    filename = ReportService.generate_pdf_report(audit_id, vulnerabilities)

    return send_file(filename, as_attachment=True, download_name=f'audit_{audit_id}.pdf')
```

Base de Datos

Esquema de Base de Datos

Tabla Users

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(80) UNIQUE NOT NULL,
    email VARCHAR(120) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabla Projects

```sql
CREATE TABLE projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    description TEXT,
    repository_url VARCHAR(500),
    user_id INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabla Audits

```sql
CREATE TABLE audits (
    id SERIAL PRIMARY KEY,
    project_id INTEGER REFERENCES projects(id),
    scan_type VARCHAR(50) DEFAULT 'all',
    status VARCHAR(20) DEFAULT 'pending',
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabla Vulnerabilities

sql

```sql
CREATE TABLE vulnerabilities (
    id SERIAL PRIMARY KEY,
    audit_id INTEGER REFERENCES audits(id),
    name VARCHAR(200) NOT NULL,
    severity VARCHAR(20) NOT NULL,
    description TEXT,
    file_path VARCHAR(1000),
    line_number INTEGER,
    cwe_id VARCHAR(20),
    recommendation TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Migraciones con Alembic

Configuración de Alembic

python

```python
# migrations/env.py
from alembic import context
from sqlalchemy import engine_from_config, pool
from app.models import *
from app.utils.database import db

config = context.config
target_metadata = db.metadata

def run_migrations_online():
    connectable = engine_from_config(
        config.get_section(config.config_ini_section),
        prefix='sqlalchemy.',
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=target_metadata
        )

        with context.begin_transaction():
            context.run_migrations()
```

Crear Migración

bash

```
# Generar nueva migración
flask db migrate -m "Descripción de la migración"

# Aplicar migración
flask db upgrade

# Revertir migración
flask db downgrade
```

Seguridad

Autenticación JWT

Configuración JWT

python

```
# config.py
import os
from datetime import timedelta

class Config:
    JWT_SECRET_KEY = os.environ.get('JWT_SECRET_KEY') or 'dev-secret-key'
    JWT_ACCESS_TOKEN_EXPIRES = timedelta(hours=24)
    JWT_REFRESH_TOKEN_EXPIRES = timedelta(days=30)
```

Middleware de Autenticación

python

```python
# app/middleware/auth_middleware.py
from flask_jwt_extended import jwt_required, get_jwt_identity
from functools import wraps

def admin_required(f):
    @wraps(f)
    @jwt_required()
    def decorated_function(*args, **kwargs):
        current_user_id = get_jwt_identity()
        user = User.query.get(current_user_id)

        if user.role != 'admin':
            return jsonify({'error': 'Admin access required'}), 403

        return f(*args, **kwargs)
    return decorated_function
```

Validación de Datos

Esquemas de Validación con Marshmallow

```python
# app/schemas/validation.py
from marshmallow import Schema, fields, validate

class ProjectSchema(Schema):
    name = fields.Str(required=True, validate=validate.Length(min=1, max=200))
    description = fields.Str(validate=validate.Length(max=1000))
    repository_url = fields.Url()

class AuditSchema(Schema):
    project_id = fields.Int(required=True)
    scan_type = fields.Str(validate=validate.OneOf(['all', 'static', 'dynamic']))
```

Protección CSRF

Configuración CSRF

```python
```

```python
# app/__init__.py
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect()

def create_app():
    app = Flask(__name__)
    csrf.init_app(app)
    return app
```

Rate Limiting

Configuración Rate Limiting

python

```python
# app/utils/rate_limit.py
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

limiter = Limiter(
    app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"]
)

# Aplicar rate limiting
@audit_bp.route('/', methods=['POST'])
@limiter.limit("10 per minute")
@jwt_required()
def create_audit():
    # ... código del endpoint
```

Testing

Configuración de Testing

Configuración PyTest

python

```python
# conftest.py
import pytest
from app import create_app
from app.utils.database import db

@pytest.fixture
def app():
    app = create_app('testing')
    with app.app_context():
        db.create_all()
        yield app
        db.session.remove()
        db.drop_all()

@pytest.fixture
def client(app):
    return app.test_client()

@pytest.fixture
def auth_headers(client):
    # Crear usuario de prueba
    response = client.post('/api/auth/register', json={
        'username': 'testuser',
        'email': 'test@example.com',
        'password': 'testpass123'
    })

    # Hacer login
    response = client.post('/api/auth/login', json={
        'username': 'testuser',
        'password': 'testpass123'
    })

    token = response.get_json()['access_token']
    return {'Authorization': f'Bearer {token}'}
```

Tests Unitarios

Test de Autenticación

python

```python
# tests/test_auth.py
def test_user_registration(client):
    response = client.post('/api/auth/register', json={
        'username': 'newuser',
        'email': 'newuser@example.com',
        'password': 'password123'
    })

    assert response.status_code == 201
    assert 'User created successfully' in response.get_json()['message']

def test_user_login(client):
    # Primero registrar usuario
    client.post('/api/auth/register', json={
        'username': 'testuser',
        'email': 'test@example.com',
        'password': 'password123'
    })

    # Intentar login
    response = client.post('/api/auth/login', json={
        'username': 'testuser',
        'password': 'password123'
    })

    assert response.status_code == 200
    assert 'access_token' in response.get_json()
```

Test de Proyectos

```python
```

```python
# tests/test_projects.py
def test_create_project(client, auth_headers):
    response = client.post('/api/projects',
        json={
            'name': 'Test Project',
            'description': 'Test Description'
        },
        headers=auth_headers
    )

    assert response.status_code == 201
    data = response.get_json()
    assert data['name'] == 'Test Project'

def test_get_projects(client, auth_headers):
    # Crear proyecto
    client.post('/api/projects',
        json={'name': 'Test Project'},
        headers=auth_headers
    )

    # Obtener proyectos
    response = client.get('/api/projects', headers=auth_headers)

    assert response.status_code == 200
    data = response.get_json()
    assert len(data) == 1
    assert data[0]['name'] == 'Test Project'
```

Tests de Integración

Test de Análisis Completo

python

```python
# tests/test_integration.py
def test_full_audit_workflow(client, auth_headers):
    # 1. Crear proyecto
    project_response = client.post('/api/projects',
        json={'name': 'Test Project'},
        headers=auth_headers
    )
    project_id = project_response.get_json()['id']

    # 2. Crear auditoría
    audit_response = client.post('/api/audits',
        json={'project_id': project_id},
        headers=auth_headers
    )
    audit_id = audit_response.get_json()['id']

    # 3. Verificar estado de auditoría
    response = client.get(f'/api/audits/{audit_id}', headers=auth_headers)
    assert response.status_code == 200

    # 4. Generar reporte
    response = client.get(f'/api/reports/{audit_id}/pdf', headers=auth_headers)
    assert response.status_code == 200
```

Tests de Rendimiento

Test de Carga

python

```python
# tests/test_performance.py
import time
import concurrent.futures

def test_concurrent_audits(client, auth_headers):
    def create_audit():
        return client.post('/api/audits',
            json={'project_id': 1},
            headers=auth_headers
        )

    # Ejecutar 10 auditorías concurrentes
    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        start_time = time.time()
        futures = [executor.submit(create_audit) for _ in range(10)]
        results = [f.result() for f in futures]
        end_time = time.time()

    # Verificar que todas las auditorías se crearon exitosamente
    assert all(r.status_code == 201 for r in results)
    assert end_time - start_time < 30  # Debe completarse en menos de 30 segundos
```

Despliegue

Despliegue con Docker

Dockerfile Backend

dockerfile

```
# Dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "run:app"]
```

Dockerfile Frontend

dockerfile

```
# frontend/Dockerfile
FROM node:16-alpine as build

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Docker Compose

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  db:
    image: postgres:14
    environment:
      POSTGRES_DB: auditoria_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  backend:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db
      - redis
    environment:
      DATABASE_URL: postgresql://postgres:password@db:5432/auditoria_db
      REDIS_URL: redis://redis:6379
    volumes:
      - ./uploads:/app/uploads

  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    depends_on:
      - backend

  worker:
    build: .
    command: celery -A app.tasks worker --loglevel=info
    depends_on:
      - db
      - redis
```

```
    environment:
      DATABASE_URL: postgresql://postgres:password@db:5432/auditoria_db
      REDIS_URL: redis://redis:6379

volumes:
  postgres_data:
```

Despliegue en Producción

Configuración Nginx

```nginx
# nginx.conf
server {
    listen 80;
    server_name tu-dominio.com;

    location / {
        proxy_pass http://frontend:80;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api {
        proxy_pass http://backend:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Variables de Entorno de Producción

```bash
# .env.production
DATABASE_URL=postgresql://user:password@db-host:5432/production_db
REDIS_URL=redis://redis-host:6379
JWT_SECRET_KEY=your-production-secret-key
FLASK_ENV=production
DEBUG=False
```

CI/CD con GitHub Actions

Pipeline de CI/CD

yaml

```yaml
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.9

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

    - name: Run tests
      run: |
        pytest --cov=app tests/
      env:
        DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db

    - name: Upload coverage
      uses: codecov/codecov-action@v1
```

```
  build:
    needs: test
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Build Docker image
      run: |
        docker build -t auditoria-web:latest .

    - name: Push to registry
      run: |
        echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin
        docker push auditoria-web:latest

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

    steps:
    - name: Deploy to production
      run: |
        # Comandos de despliegue
        echo "Deploying to production..."
```

Mantenimiento

Monitoring y Logging

Configuración de Logging

```
python
```

```python
# app/utils/logging.py
import logging
from logging.handlers import RotatingFileHandler
import os

def setup_logging(app):
    if not app.debug:
        if not os.path.exists('logs'):
            os.mkdir('logs')

        file_handler = RotatingFileHandler(
            'logs/auditoria.log',
            maxBytes=10240000,
            backupCount=10
        )
        file_handler.setFormatter(logging.Formatter(
            '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]'
        ))
        file_handler.setLevel(logging.INFO)
        app.logger.addHandler(file_handler)

        app.logger.setLevel(logging.INFO)
        app.logger.info('Auditoria Web startup')
```

Métricas con Prometheus

```
python
```

```python
# app/utils/metrics.py
from prometheus_client import Counter, Histogram, generate_latest
import time

# Métricas
REQUEST_COUNT = Counter('requests_total', 'Total requests', ['method', 'endpoint'])
REQUEST_DURATION = Histogram('request_duration_seconds', 'Request duration')

def track_requests(f):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        REQUEST_COUNT.labels(method=request.method, endpoint=request.endpoint).inc()

        result = f(*args, **kwargs)

        REQUEST_DURATION.observe(time.time() - start_time)
        return result
    return wrapper

@app.route('/metrics')
def metrics():
    return generate_latest()
```

Backup y Recuperación

Script de Backup

python

```python
# scripts/backup.py
import subprocess
import os
from datetime import datetime

def backup_database():
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    backup_file = f"backup_{timestamp}.sql"

    cmd = [
        'pg_dump',
        '-h', 'localhost',
        '-U', 'postgres',
        '-d', 'auditoria_db',
        '-f', backup_file
    ]

    try:
        subprocess.run(cmd, check=True)
        print(f"Backup created: {backup_file}")

        # Comprimir backup
        subprocess.run(['gzip', backup_file], check=True)
        print(f"Backup compressed: {backup_file}.gz")

    except subprocess.CalledProcessError as e:
        print(f"Error creating backup: {e}")

if __name__ == "__main__":
    backup_database()
```

Actualización y Migración

Script de Actualización

```
python
```

```python
# scripts/update.py
import subprocess
import sys

def update_application():
    steps = [
        ("Updating code", ["git", "pull", "origin", "main"]),
        ("Installing dependencies", ["pip", "install", "-r", "requirements.txt"]),
        ("Running migrations", ["flask", "db", "upgrade"]),
        ("Collecting static files", ["python", "manage.py", "collectstatic"]),
        ("Restarting services", ["systemctl", "restart", "auditoria-web"])
    ]

    for description, command in steps:
        print(f"Step: {description}")
        try:
            subprocess.run(command, check=True)
            print("✓ Success")
        except subprocess.CalledProcessError as e:
            print(f"✗ Error: {e}")
            sys.exit(1)

    print("Application updated successfully!")

if __name__ == "__main__":
    update_application()
```

Documentación de API

Swagger/OpenAPI

Configuración Swagger

python

```python
# app/utils/swagger.py
from flask_restx import Api, Resource, fields
from flask import Blueprint

api_bp = Blueprint('api', __name__)
api = Api(api_bp, version='1.0', title='Auditoria Web API',
          description='API para el sistema de auditoría de seguridad web')

# Modelos
user_model = api.model('User', {
    'id': fields.Integer(required=True, description='User ID'),
    'username': fields.String(required=True, description='Username'),
    'email': fields.String(required=True, description='Email address'),
    'role': fields.String(description='User role')
})

project_model = api.model('Project', {
    'id': fields.Integer(required=True, description='Project ID'),
    'name': fields.String(required=True, description='Project name'),
    'description': fields.String(description='Project description'),
    'repository_url': fields.String(description='Repository URL')
})
```

Endpoints Documentados

Ejemplo de Endpoint Documentado

python

```python
@api.route('/projects')
class ProjectList(Resource):
    @api.doc('list_projects')
    @api.marshal_list_with(project_model)
    @jwt_required()
    def get(self):
        """Fetch all projects for authenticated user"""
        user_id = get_jwt_identity()
        projects = Project.query.filter_by(user_id=user_id).all()
        return projects

    @api.doc('create_project')
    @api.expect(project_model)
    @api.marshal_with(project_model, code=201)
    @jwt_required()
    def post(self):
        """Create a new project"""
        user_id = get_jwt_identity()
        data = api.payload

        project = Project(
            name=data['name'],
            description=data.get('description', ''),
            user_id=user_id
        )

        db.session.add(project)
        db.session.commit()

        return project, 201
```

Contacto y Soporte

Información del Equipo de Desarrollo

- **Repositorio**: https://github.com/usuario/sistema-auditoria-web
- **Documentación**: https://docs.auditoria-web.com
- **Issues**: https://github.com/usuario/sistema-auditoria-web/issues

Contribuir al Proyecto

1. Fork el repositorio
2. Crear una rama para tu feature (`git checkout -b feature/nueva-funcionalidad`)
3. Commit los cambios (`git commit -am 'Agregar nueva funcionalidad'`)

4. Push a la rama (`git push origin feature/nueva-funcionalidad`)
5. Crear un Pull Request

Licencia

Este proyecto está bajo la licencia MIT. Ver el archivo `LICENSE` para más detalles.

Este manual técnico ha sido desarrollado para facilitar el desarrollo y mantenimiento del Sistema de Auditoría de Seguridad Web. Mantén esta documentación actualizada con cada cambio significativo en el código.