JIRA BACKLOG - SENIAL MODERNIZATION

Tickets Listos para Importación

🎯 EPIC: INFRAESTRUCTURA Y SEGURIDAD

Epic Key: `EPIC-INFRA`
Epic Name: Infraestructura y Seguridad Crítica
Epic Owner: Tech Lead
Business Value: Eliminar vulnerabilidades críticas y establecer base moderna

---

TICKET: SENIAL-001

```
Summary: Modernizar versión de Python de 3.4 a 3.11 LTS
Issue Type: Task
Priority: Critical
Sprint: Sprint 1
Story Points: 5
Epic Link: EPIC-INFRA
Assignee: Senior Developer
Labels: python, migration, critical, infrastructure

Description:
Migrar el proyecto completo de Python 3.4 (EOL 2019) a Python 3.11 LTS para resolver vulnerabilidades de
seguridad y obtener soporte a largo plazo.

CONTEXTO:
- Python 3.4 sin soporte desde 2019
- Múltiples vulnerabilidades de seguridad conocidas
- Incompatibilidades con librerías modernas

SCOPE:
- Instalación y configuración Python 3.11
- Actualización de sintaxis deprecated
- Verificación de compatibilidad de dependencias
- Testing completo de funcionalidad

Acceptance Criteria:
- [ ] Python 3.11+ instalado en todos los ambientes
- [ ] Código ejecutándose sin errores en nueva versión
- [ ] Todas las funcionalidades core operativas
- [ ] Performance igual o mejor que versión anterior
- [ ] Documentación de migración creada

Technical Tasks:
- [ ] Setup Python 3.11 en desarrollo
- [ ] Auditar código para breaking changes
- [ ] Actualizar f-strings y sintaxis moderna
- [ ] Verificar imports y módulos deprecated
- [ ] Ejecutar test suite completo
- [ ] Update README con nueva versión

Definition of Done:
- [ ] Code review aprobado
- [ ] Tests pasando en Python 3.11
- [ ] No errores ni warnings
- [ ] Documentación actualizada

Blocked by: N/A
Blocks: SENIAL-002, SENIAL-005

Time Tracking:
Original Estimate: 20h
Remaining Estimate: 20h
```

TICKET: SENIAL-002

```
Summary: Implementar gestión moderna de dependencias con pyproject.toml
Issue Type: Task
Priority: High
Sprint: Sprint 1
Story Points: 3
Epic Link: EPIC-INFRA
Assignee: Developer
Labels: dependencies, pyproject, packaging

Description:
Crear un sistema de gestión de dependencias moderno usando pyproject.toml y requirements files
estructurados para mejorar la reproducibilidad y el mantenimiento.

PROBLEMA ACTUAL:
- Sin gestión formal de dependencias
- Instalación manual de librerías
- Versiones no controladas
- Difficulty en setup de nuevos ambientes

SOLUCIÓN:
```

- pyproject.toml con metadata del proyecto
- requirements.txt para producción
- requirements-dev.txt para desarrollo
- Scripts de setup automatizado

Acceptance Criteria:
- [ ] pyproject.toml creado con metadata completa
- [ ] requirements.txt con dependencias de producción
- [ ] requirements-dev.txt con herramientas de desarrollo
- [ ] Script de setup one-command funcional
- [ ] Documentación de instalación actualizada

Technical Tasks:
- [ ] Crear pyproject.toml con build-system
- [ ] Catalogar todas las dependencias actuales
- [ ] Separar deps de producción vs desarrollo
- [ ] Crear requirements files versionados
- [ ] Script setup.py o Makefile para instalación
- [ ] Test en ambiente limpio

Files to Create:
- pyproject.toml
- requirements.txt
- requirements-dev.txt
- scripts/setup.sh

Depends on: SENIAL-001

TICKET: SENIAL-003

Summary: [SECURITY] Resolver vulnerabilidad SECRET_KEY hardcodeada
Issue Type: Bug
Priority: Critical
Sprint: Sprint 1
Story Points: 2
Epic Link: EPIC-INFRA
Assignee: Tech Lead
Labels: security, critical, configuration, vulnerability

Description:
VULNERABILIDAD CRÍTICA: SECRET_KEY está hardcodeada en el código fuente lo que presenta un riesgo de seguridad alto. Debe externalizarse inmediatamente.

AFFECTED FILES:
- 01_presentacion/webapp/flask_main.py:14

~~~
- 01_presentacion/webapp/views.py:8

CURRENT CODE:
```python
app.config['SECRET_KEY'] = "Victor"
~~~

SECURITY IMPACT:

- Session hijacking possible
- CSRF attacks enabled
- Predictable encryption keys
- Code repository exposure

SOLUTION:

- Environment variables for secrets
- .env files for development
- Secret management for production
- Input validation and fallbacks

Acceptance Criteria:

☐ SECRET_KEY removed from all source files
☐ Environment variable loading implemented
☐ .env file created for development
☐ .env.example provided for team
☐ Validation of required config at startup
☐ No secrets in git history

Technical Implementation:

☐ Install python-dotenv
☐ Create config.py module for settings
☐ Environment variable loading
☐ Startup validation of required config
☐ Update deployment documentation
☐ Security audit of other hardcoded values

Security Checklist:

☐ No secrets in source code
☐ .env in .gitignore
☐ Strong default generation for secrets
☐ Configuration validation at startup

- [ ] Documentation for secret rotation

Time Critical: Must be completed in Week 1
Risk Level: HIGH if not addressed immediately

```
---

### TICKET: SENIAL-004
```

Summary: Externalizar configuración hardcodeada a sistema flexible
Issue Type: Improvement
Priority: High
Sprint: Sprint 1
Story Points: 3
Epic Link: EPIC-INFRA
Assignee: Developer
Labels: configuration, refactoring, portability

Description:
Migrar la configuración basada en paths absolutos y valores hardcodeados a un sistema de configuración flexible y portable.

CURRENT ISSUES:

- Paths absolutos en XML: /Users/victor/PycharmProjects/DDD/
- Configuración no portable entre ambientes
- Duplicación de configuración
- Sin validación de configuración

FILES AFFECTED:

- 03_aplicacion/datos/configuracion.xml
- 01_presentacion/webapp/datos/configuracion.xml
- 03_aplicacion/contenedor/configurador.py

SOLUTION APPROACH:

- YAML/TOML configuration files
- Environment-specific configurations
- Configuration schema validation
- Relative paths with base directory

Acceptance Criteria:

- [ ] Configuration externalized to YAML/TOML
- [ ] Environment variables for paths
- [ ] Schema validation implemented
- [ ] Multiple environment support (dev/test/prod)
- [ ] Backward compatibility maintained
- [ ] Migration documentation

Technical Tasks:

- [ ] Design configuration schema
- [ ] Implement configuration loader
- [ ] Replace XML parsing with YAML/TOML
- [ ] Add environment variable override
- [ ] Implement validation with pydantic/cerberus
- [ ] Create configs for different environments
- [ ] Update configurador.py to use new system

Configuration Structure:

```yaml
app:
  name: SenialSOLID
  version: "2.0"

paths:
  data_dir: ${DATA_DIR:-./data}
  acquisition_dir: ${ACQ_DIR:-./data/adq}
  processing_dir: ${PROC_DIR:-./data/pro}

acquisition:
  type: senoidal
  input_file: ${INPUT_FILE:-datos.txt}

processing:
  type: umbral
  threshold: ${THRESHOLD:-5}

signals:
  acquisition_type: pila
  processing_type: pila
  size: ${SIGNAL_SIZE:-20}
```

Depends on: SENIAL-003

```
---

## 🌐 EPIC: MODERNIZACIÓN WEB
**Epic Key:** `EPIC-WEB`
**Epic Name:** Modernización Framework Web
**Epic Owner:** Senior Developer
**Business Value:** Framework web moderno y mantenible

---

### TICKET: SENIAL-005
```

Summary: Actualizar Flask y extensiones obsoletas (flask.ext.*) Issue Type: Task Priority: High Sprint: Sprint 2 Story Points: 8 Epic Link: EPIC-WEB Assignee: Senior Developer Labels: flask, modernization, web, breaking-changes

Description:
Migrar de Flask con extensiones obsoletas (flask.ext.*) a versiones modernas compatibles con Flask 2.x+

CURRENT PROBLEMATIC IMPORTS:

```python
from flask.ext.bootstrap import Bootstrap      # DEPRECATED
from flask.ext.moment import Moment          # DEPRECATED
from flask.ext.wtf import Form              # DEPRECATED
from flask.ext.sqlalchemy import SQLAlchemy # DEPRECATED
```

TARGET MODERN IMPORTS:

```python
from flask_bootstrap import Bootstrap
from flask_moment import Moment
from flask_wtf import FlaskForm
from flask_sqlalchemy import SQLAlchemy
```

MIGRATION COMPLEXITY:

- API changes in extensions
- Breaking changes in Flask 2.x
- Template compatibility issues

- Form handling updates

Acceptance Criteria:

- [ ] Flask upgraded to 2.3+
- [ ] All flask.ext.* imports replaced
- [ ] Flask-Bootstrap working with modern templates
- [ ] Flask-WTF forms functioning correctly
- [ ] Flask-SQLAlchemy database operations working
- [ ] All web routes responding correctly
- [ ] No deprecation warnings

Technical Tasks:

- [ ] Update requirements with modern Flask versions
- [ ] Replace all flask.ext imports
- [ ] Update Form classes to FlaskForm
- [ ] Test all web endpoints
- [ ] Verify template rendering
- [ ] Update error handlers (404, 500)
- [ ] Test form submissions and validation

Files to Update:

- 01_presentacion/webapp/flask_main.py
- 01_presentacion/webapp/views.py
- 01_presentacion/webapp/forms.py
- All HTML templates

Testing Priority:

- [ ] Home page loads
- [ ] Navigation works
- [ ] Forms submit correctly
- [ ] Error pages display
- [ ] Bootstrap styling intact

Breaking Changes Documentation:

- Form → FlaskForm migration
- Template context changes
- Error handling updates

Depends on: SENIAL-001, SENIAL-002
Blocks: SENIAL-006

```
---

### TICKET: SENIAL-006
```

Summary: Modernizar templates y mejorar responsive design
Issue Type: Improvement Priority: Medium Sprint: Sprint 2 Story Points: 5 Epic Link: EPIC-WEB
Assignee: Developer Labels: ui, bootstrap, responsive, templates

Description:
Actualizar templates HTML para usar Bootstrap 5 y mejorar la experiencia responsive en dispositivos móviles.

CURRENT STATE:

- Bootstrap 3.x (obsoleto)
- Templates no optimizados para mobile
- Formularios sin validación client-side
- Navegación básica

MODERNIZATION GOALS:

- Bootstrap 5.x con utilidades modernas
- Mobile-first responsive design
- Client-side form validation
- Improved navigation UX
- Modern CSS Grid/Flexbox

Acceptance Criteria:

- [ ] Bootstrap 5.x integrado correctamente
- [ ] Todos los templates responsive verificados
- [ ] Formularios con validación client-side
- [ ] Navegación mejorada y accesible
- [ ] Cross-browser compatibility testing
- [ ] Performance de carga optimizada

Technical Tasks:

- [ ] Upgrade Bootstrap CDN to 5.x

- [ ] Update base.html template structure
- [ ] Modernize navigation component
- [ ] Add responsive breakpoints
- [ ] Implement client-side validation
- [ ] Optimize CSS loading
- [ ] Test on multiple devices/browsers

Templates to Update:

- templates/general/base.html
- templates/aplicacion/adquisicion.html
- templates/aplicacion/procesamiento.html
- templates/aplicacion/visualizacion.html
- All form templates

Mobile Testing Checklist:

- [ ] iPhone (Safari)
- [ ] Android (Chrome)
- [ ] Tablet views
- [ ] Desktop responsive
- [ ] Accessibility (ARIA labels)

Depends on: SENIAL-005

```

---

## 🔧 EPIC: CALIDAD DE CÓDIGO
**Epic Key:** `EPIC-QUALITY`
**Epic Name:** Mejoras de Calidad y Mantenibilidad
**Epic Owner:** Tech Lead
**Business Value:** Código mantenible y observable

---

### TICKET: SENIAL-007
```

Summary: Implementar sistema de logging estructurado
Issue Type: Improvement
Priority: Medium
Sprint: Sprint 3
Story Points: 8

Epic Link: EPIC-QUALITY
Assignee: Senior Developer
Labels: logging, observability, monitoring

Description:
Reemplazar todas las declaraciones print() con un sistema de logging profesional y estructurado para mejorar la observabilidad y debugging.

CURRENT PROBLEMS:

- print() statements throughout codebase
- No log levels or categorization
- No centralized logging configuration
- Difficult to debug production issues
- No log rotation or management

LOGGING STRATEGY:

- Structured JSON logging for production
- Multiple log levels (DEBUG, INFO, WARNING, ERROR, CRITICAL)
- Centralized configuration
- Correlation IDs for request tracing
- Log rotation and retention policies
- Environment-specific log levels

TARGET ARCHITECTURE:

python

```
import logging
import structlog

# Structured logging setup
structlog.configure(
    processors=[
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.processors.add_log_level,
        structlog.processors.JSONRenderer()
    ],
    wrapper_class=structlog.make_filtering_bound_logger(logging.INFO),
    logger_factory=structlog.WriteLoggerFactory(),
    cache_logger_on_first_use=True,
)

logger = structlog.get_logger(__name__)
logger.info("Signal acquired", signal_id=123, values_count=20)
```

Acceptance Criteria:

- [ ] All print() statements replaced with appropriate logging
- [ ] Centralized logging configuration implemented
- [ ] JSON structured logging for production
- [ ] Log levels properly assigned (DEBUG/INFO/WARNING/ERROR)
- [ ] Correlation IDs for request tracing
- [ ] Log rotation configured
- [ ] Environment-specific log levels

Technical Tasks:

- [ ] Install and configure structlog/loguru
- [ ] Create centralized logging configuration
- [ ] Replace print statements in all modules
- [ ] Implement correlation ID middleware for Flask
- [ ] Configure log rotation (logrotate/TimedRotatingFileHandler)
- [ ] Add log level configuration per environment
- [ ] Create logging documentation/guidelines

Files to Modify:

- All .py files with print() statements
- New: logging_config.py
- New: middleware/correlation.py
- Update: requirements.txt

Logging Categories:

- app.acquisition: Signal acquisition events
- app.processing: Signal processing events
- app.web: Web request/response
- app.persistence: Database/file operations
- app.config: Configuration loading
- app.security: Authentication/authorization

Performance Considerations:

- Async logging for high-throughput
- Log sampling for DEBUG level
- Structured data serialization optimization

Depends on: SENIAL-001

```
---

### TICKET: SENIAL-008
```

Summary: Refactorizar manejo genérico de excepciones Issue Type: Improvement Priority: Medium
Sprint: Sprint 3
Story Points: 5 Epic Link: EPIC-QUALITY Assignee: Developer Labels: exceptions, error-handling, reliability

Description:
Implementar manejo específico de excepciones en lugar del manejo genérico actual para mejorar debugging y user experience.

CURRENT ISSUES:

- Generic Exception handling masks specific errors
- print() statements for error reporting
- No user-friendly error messages
- Difficult debugging due to generic catching
- No error categorization or recovery strategies

PROBLEMATIC PATTERNS:

```python
```

```
try:
    # some operation
    pass
except Exception as ex:
    print("Error: " + str(ex))  # Too generic!
    raise ex
```

TARGET PATTERN:

```python
try:
    # some operation
    pass
except FileNotFoundError as e:
    logger.error("Configuration file not found", path=config_path, error=str(e))
    raise ConfigurationError(f"Required configuration file missing: {config_path}")
except PermissionError as e:
    logger.error("Permission denied accessing file", path=config_path, error=str(e))
    raise ConfigurationError(f"Permission denied: {config_path}")
except ValueError as e:
    logger.error("Invalid configuration format", error=str(e))
    raise ConfigurationError(f"Invalid configuration format: {e}")
```

Acceptance Criteria:

- [ ] Custom exception classes for domain-specific errors
- [ ] Specific exception handling instead of generic Exception
- [ ] User-friendly error messages for web interface
- [ ] Proper error logging with context
- [ ] Error recovery strategies where possible
- [ ] HTTP error handlers for web endpoints

Technical Tasks:

- [ ] Create custom exception hierarchy
- [ ] Replace generic Exception catching
- [ ] Implement Flask error handlers
- [ ] Add contextual error logging
- [ ] Create user-friendly error messages
- [ ] Test error scenarios and recovery

Custom Exceptions to Create:

```python
```

```
class SenialError(Exception):
    """Base exception for Senial application"""
    pass

class ConfigurationError(SenialError):
    """Configuration related errors"""
    pass

class SignalProcessingError(SenialError):
    """Signal processing related errors"""
    pass

class PersistenceError(SenialError):
    """Data persistence related errors"""
    pass

class ValidationError(SenialError):
    """Input validation errors"""
    pass
```

Flask Error Handlers:

- 400: Bad Request (validation errors)
- 404: Not Found (custom page)
- 500: Internal Server Error (logged with correlation ID)
- Custom: Business logic errors

Files to Modify:

- All controllers and managers
- Flask app error handlers
- Repository classes
- Processing modules

Testing Strategy:

- Unit tests for each exception type
- Error scenario testing
- User experience testing for error pages

Depends on: SENIAL-007

```
---

### TICKET: SENIAL-009
```

Summary: Implementar validación robusta de entrada de datos Issue Type: Improvement
Priority: Medium Sprint: Sprint 3 Story Points: 5 Epic Link: EPIC-QUALITY Assignee: Developer Labels: validation, security, input-sanitization

Description:
Añadir validación y sanitización completa para todas las entradas de usuario para mejorar seguridad y confiabilidad.

CURRENT VULNERABILITIES:

- No input validation on forms
- File uploads without validation
- Configuration files without schema validation
- SQL injection potential (if added DB queries)
- XSS vulnerabilities in templates

VALIDATION LAYERS:

1. Client-side: JavaScript form validation (UX)
2. Server-side: Python input validation (Security)
3. Database: Schema constraints (Data integrity)
4. Business: Domain rule validation (Logic)

TARGET IMPLEMENTATION:

```
python
```

```python
from pydantic import BaseModel, validator, Field
from marshmallow import Schema, fields, validate

class SignalAcquisitionRequest(BaseModel):
    identificador: int = Field(..., ge=1, le=9999, description="Signal ID")
    descripcion: str = Field(..., min_length=1, max_length=255)
    fecha: date = Field(...)

    @validator('descripcion')
    def validate_description(cls, v):
        # Sanitize HTML and validate content
        return bleach.clean(v.strip())
```

Acceptance Criteria:

- [ ] All web forms with client-side validation
- [ ] Server-side validation for all endpoints
- [ ] Input sanitization to prevent XSS
- [ ] File upload validation (type, size, content)
- [ ] Configuration schema validation
- [ ] Rate limiting for API endpoints

Technical Tasks:

- [ ] Implement client-side validation with JavaScript
- [ ] Add server-side validation with Pydantic/Marshmallow
- [ ] Install and configure input sanitization (bleach)
- [ ] Create validation schemas for all forms
- [ ] Add CSRF protection verification
- [ ] Implement rate limiting middleware
- [ ] Add file upload validation

Validation Rules:

- Signal ID: Integer, range 1-9999
- Description: String, 1-255 chars, no HTML
- Date: Valid date format, not future
- File uploads: Max 10MB, allowed types only
- Configuration: Valid YAML/JSON schema

Security Measures:

- [ ] HTML sanitization with bleach
- [ ] CSRF token validation

- [ ] Rate limiting (10 req/min per IP)
- [ ] File type validation (magic bytes)
- [ ] Input length limits
- [ ] SQL injection prevention (parameterized queries)

Files to Create/Modify:

- validators/schemas.py (validation schemas)
- middleware/validation.py (validation middleware)
- static/js/validation.js (client-side validation)
- templates/ (add validation feedback)

Testing Requirements:

- [ ] Valid input acceptance tests
- [ ] Invalid input rejection tests
- [ ] XSS attempt prevention tests
- [ ] File upload security tests
- [ ] Rate limiting tests

Depends on: SENIAL-008

```


---


## 🧪 EPIC: TESTING Y AUTOMATIZACIÓN
**Epic Key:** `EPIC-TEST`
**Epic Name:** Testing y CI/CD Pipeline
**Epic Owner:** QA Engineer
**Business Value:** Calidad asegurada y deployment automático


---


### TICKET: SENIAL-010
```

Summary: Implementar suite completa de pruebas unitarias
Issue Type: Task
Priority: High
Sprint: Sprint 4
Story Points: 13
Epic Link: EPIC-TEST
Assignee: QA Engineer + Senior Developer
Labels: testing, pytest, coverage, quality

Description:
Crear una suite completa de pruebas unitarias con cobertura mínima del 80% para asegurar la calidad y facilitar futuras refactorizaciones.

CURRENT STATE:

- 0% test coverage
- No automated testing
- Manual testing only
- Risk of regressions with changes

TESTING STRATEGY:

- Unit tests for domain logic (models, processors)
- Integration tests for controllers
- Component tests for web endpoints
- Fixture-based test data management
- Mocking external dependencies

TARGET ARCHITECTURE:

```python
# pytest configuration
# pytest.ini
[tool:pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
python_functions = test_*
addopts = --cov=. --cov-report=html --cov-report=term --cov-fail-under=80

# Example test structure
class TestSignalProcessing:
    def test_signal_amplification(self, sample_signal):
        processor = Procesador(output_signal)
        processor.procesar(sample_signal)
        result = processor.obtener_senial_procesada()

        assert result.cantidad == sample_signal.cantidad
        for i in range(result.cantidad):
            assert result.obtener_valor(i) == sample_signal.obtener_valor(i) * 2
```

Acceptance Criteria:

- [ ] pytest configured with all necessary plugins
- [ ] 80%+ code coverage achieved
- [ ] Unit tests for all domain models (Senial, Procesador, etc.)
- [ ] Integration tests for controllers
- [ ] Web endpoint testing with test client
- [ ] Fixtures for test data management
- [ ] Mocking for external dependencies (files, config)
- [ ] Continuous testing workflow

Technical Tasks:

- [ ] Install and configure pytest + plugins
- [ ] Create test directory structure
- [ ] Write fixtures for test data
- [ ] Unit tests for domain models:
- [ ] test_senial.py (all signal types)
- [ ] test_procesador.py (all processors)
- [ ] test_adquisidor.py (all acquisitors)
- [ ] Integration tests for managers:
- [ ] test_controlador_adquisicion.py
- [ ] test_controlador_procesamiento.py
- [ ] Web tests:
- [ ] test_views.py (all endpoints)
- [ ] test_forms.py (form validation)
- [ ] Repository tests with temporary files
- [ ] Configuration tests

Test Structure:

```
tests/
├── unit/
│   ├── domain/
│   │   ├── test_senial.py
│   │   ├── test_procesador.py
│   │   └── test_adquisidor.py
│   ├── managers/
│   │   ├── test_controlador_adquisicion.py
│   │   └── test_controlador_procesamiento.py
│   └── repositories/
│       └── test_repositorio.py
├── integration/
│   ├── test_end_to_end.py
│   └── test_web_endpoints.py
├── fixtures/
│   ├── conftest.py
│   └── test_data.py
└── utils/
    └── test_helpers.py
```

Coverage Targets by Module:

- Domain models: 90%+
- Controllers: 85%+
- Web views: 80%+
- Repositories: 85%+
- Configuration: 75%+

Testing Tools:

- pytest: Test runner
- pytest-cov: Coverage reporting
- pytest-mock: Mocking framework
- pytest-flask: Flask testing utilities
- factory-boy: Test data factories
- pytest-xdist: Parallel test execution

Mock Strategy:

- File system operations (tempdir)
- Configuration loading (mock config)
- External dependencies

- Time-dependent operations

Performance Testing:

- [ ] Load testing for web endpoints
- [ ] Memory usage validation
- [ ] Processing time benchmarks

Depends on: SENIAL-007, SENIAL-008, SENIAL-009

```


---

### TICKET: SENIAL-011
```

Summary: Configurar CI/CD pipeline con GitHub Actions Issue Type: Task Priority: Medium Sprint: Sprint 4 Story Points: 8
Epic Link: EPIC-TEST Assignee: DevOps + Tech Lead Labels: cicd, github-actions, automation, deployment

Description:
Implementar pipeline de CI/CD completo con GitHub Actions para automatizar testing, quality checks y deployment.

PIPELINE GOALS:

- Automated testing on every PR
- Code quality enforcement
- Security vulnerability scanning
- Automated deployment to staging
- Release management automation

WORKFLOW STAGES:

1. **CI Pipeline** (on PR):
   - Linting and formatting
   - Unit and integration tests
   - Security scanning
   - Code quality metrics
2. **CD Pipeline** (on merge to main):
   - Build and package application
   - Deploy to staging environment

- Run smoke tests
- Optional: Deploy to production

TARGET WORKFLOW:

```yaml
```

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

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

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.11, 3.12]

    steps:
    - uses: actions/checkout@v4
    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: ${{ matrix.python-version }}

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

    - name: Lint with pylint
      run: pylint **/*.py

    - name: Format check with black
      run: black --check .

    - name: Security scan with bandit
      run: bandit -r . -f json

    - name: Run tests
      run: pytest --cov=. --cov-report=xml

    - name: Upload coverage
      uses: codecov/codecov-action@v3
```

Acceptance Criteria:

- [ ] GitHub Actions workflows configured

- [ ] CI pipeline running on every PR
- [ ] All quality checks automated (lint, format, security)
- [ ] Test results reported in PRs
- [ ] Coverage reports generated and tracked
- [ ] CD pipeline deploying to staging
- [ ] Notification system for failures
- [ ] Release automation configured

Technical Implementation:

- [ ] Create .github/workflows/ci.yml
- [ ] Configure Python matrix testing (3.11+)
- [ ] Set up code quality checks:
- [ ] pylint for code quality
- [ ] black for formatting
- [ ] isort for import sorting
- [ ] bandit for security scanning
- [ ] Configure test execution and reporting
- [ ] Set up coverage reporting (codecov)
- [ ] Create staging deployment workflow
- [ ] Configure secrets management
- [ ] Set up notification webhooks

Quality Gates:

- [ ] All tests must pass
- [ ] Coverage must be >80%
- [ ] Pylint score must be >8.0
- [ ] No critical security vulnerabilities
- [ ] Code must be formatted with black

Deployment Strategy:

- **Staging**: Auto-deploy on merge to main
- **Production**: Manual approval required
- **Rollback**: Automated rollback on health check failure

Environments:

```yaml
```

```yaml
environments:
  staging:
    url: https://senial-staging.herokuapp.com
    variables:
      FLASK_ENV: staging
      DATABASE_URL: ${{ secrets.STAGING_DB_URL }}

  production:
    url: https://senial-app.herokuapp.com
    protection_rules:
      required_reviewers: 1
    variables:
      FLASK_ENV: production
      DATABASE_URL: ${{ secrets.PROD_DB_URL }}
```

Monitoring and Alerts:

- [ ] Slack notifications for build failures
- [ ] Email alerts for deployment issues
- [ ] Status badges in README
- [ ] Performance regression detection

Security Considerations:

- [ ] Secrets managed via GitHub Secrets
- [ ] No sensitive data in logs
- [ ] Dependency vulnerability scanning
- [ ] SAST (Static Application Security Testing)

Documentation:

- [ ] CI/CD setup documentation
- [ ] Deployment runbooks
- [ ] Troubleshooting guides
- [ ] Release process documentation

Depends on: SENIAL-010

```
---

## 📚 EPIC: DOCUMENTACIÓN Y UX
**Epic Key:** `EPIC-DOC`
**Epic Name:** Documentación y Experiencia de Usuario
**Epic Owner:** Tech Lead
**Business Value:** Producto profesional y bien documentado

---

### TICKET: SENIAL-012
```

Summary: Actualizar documentación técnica completa del proyecto
Issue Type: Task
Priority: Medium
Sprint: Sprint 5
Story Points: 5
Epic Link: EPIC-DOC
Assignee: Tech Lead + Developer
Labels: documentation, readme, architecture

Description:
Crear documentación técnica completa y actualizada que refleje el estado modernizado del proyecto y facilite la colaboración del equipo.

CURRENT DOCUMENTATION STATE:

- README básico y desactualizado
- Sin documentación de arquitectura
- Sin guías de contribución
- Sin documentación de API
- Sin runbooks de deployment

DOCUMENTATION STRATEGY:

- Living documentation that stays current
- Multiple formats for different audiences
- Automated documentation generation where possible
- Clear separation between user and technical docs

TARGET DOCUMENTATION STRUCTURE:

```
docs/
├── README.md (updated)
├── CONTRIBUTING.md
├── ARCHITECTURE.md
├── API.md
├── DEPLOYMENT.md
├── TROUBLESHOOTING.md
├── CHANGELOG.md
└── assets/
    ├── architecture-diagram.png
    └── screenshots/
```

Acceptance Criteria:

- [ ] README.md completamente actualizado con setup moderno
- [ ] Documentación de arquitectura con diagramas
- [ ] Guía de contribución para nuevos desarrolladores
- [ ] Documentación de API endpoints
- [ ] Runbooks de deployment y troubleshooting
- [ ] Changelog detallado desde modernización
- [ ] Screenshots actualizados de la aplicación

Technical Tasks:

- [ ] Update README.md:
- [ ] Modern Python setup instructions
- [ ] Dependencies installation
- [ ] Local development setup
- [ ] Testing instructions
- [ ] Contributing guidelines
- [ ] Create ARCHITECTURE.md:
- [ ] System overview and layers
- [ ] SOLID principles implementation
- [ ] Design patterns used
- [ ] Data flow diagrams
- [ ] Technology stack documentation
- [ ] Create API.md:
- [ ] REST endpoints documentation
- [ ] Request/response examples
- [ ] Error codes and handling
- [ ] Authentication (if applicable)
- [ ] Create operational docs:

- [ ] DEPLOYMENT.md with step-by-step guides
- [ ] TROUBLESHOOTING.md for common issues
- [ ] Environment setup guides
- [ ] Generate visual documentation:
- [ ] Architecture diagrams (draw.io/mermaid)
- [ ] Application screenshots
- [ ] Flow diagrams for processes

README.md Structure:

~~~markdown
# SenialSOLID - Modern Signal Processing Application

## 🎯 Overview
Brief description and key features

## 🏗 Architecture
High-level architecture overview

## 🚀 Quick Start
```bash
# One-command setup
make install && make run
~~~

📋 Requirements

- Python 3.11+
- Dependencies listed in requirements.txt

🛠 Development

Local development setup and guidelines

🧪 Testing

How to run tests and coverage

🚢 Deployment

Production deployment instructions

🤝 Contributing

Guidelines for contributors

📄 License

License information

```
Documentation Standards:
- [ ] Markdown format for consistency
- [ ] Code examples with syntax highlighting
- [ ] Screenshots for visual components
- [ ] Diagrams for complex concepts
- [ ] Links to external resources
- [ ] Version information in all docs

Automation:
- [ ] Auto-generate API docs from code
- [ ] Keep changelog updated via CI
- [ ] Link checking in documentation
- [ ] Screenshot automation for UI changes

Quality Checklist:
- [ ] All setup instructions tested on clean environment
- [ ] Code examples verified and working
- [ ] Screenshots current and high-quality
- [ ] Links verified and working
- [ ] Grammar and spelling checked

Depends on: All previous tickets (documentation reflects final state)
```

TICKET: SENIAL-013

```
Summary: Mejorar experiencia de usuario y pulir interfaz web
Issue Type: Improvement
Priority: Low
Sprint: Sprint 5
Story Points: 8
Epic Link: EPIC-DOC
Assignee: Developer
Labels: ux, ui, polish, usability

Description:
Implementar mejoras de experiencia de usuario para hacer la aplicación más intuitiva y profesional.

CURRENT UX ISSUES:
- No feedback visual para acciones del usuario
```

~~~
- Estados de carga no indicados
- Mensajes de error técnicos y poco amigables
- Navegación básica sin breadcrumbs
- Sin indicadores de progreso
- Formularios sin ayuda contextual

UX IMPROVEMENT AREAS:
1. **Visual Feedback**: Loading states, success/error messages
2. **Navigation**: Breadcrumbs, active states, clear hierarchy
3. **Forms**: Inline validation, help text, progress indicators
4. **Accessibility**: ARIA labels, keyboard navigation, contrast
5. **Performance**: Perceived performance improvements

TARGET UX ENHANCEMENTS:
```javascript
// Loading states example
const showLoading = (element) => {
  element.innerHTML = `
    <div class="spinner-border spinner-border-sm" role="status">
      <span class="visually-hidden">Cargando...</span>
    </div> Procesando señal...
  `;
};

// Success feedback
const showSuccess = (message) => {
  toastr.success(message, 'Operación exitosa', {
    progressBar: true,
    timeOut: 3000
  });
};
~~~

Acceptance Criteria:

- [ ] Loading states para todas las operaciones largas
- [ ] Feedback visual inmediato para acciones del usuario
- [ ] Mensajes de error user-friendly con sugerencias
- [ ] Navegación mejorada con breadcrumbs
- [ ] Formularios con validación inline y ayuda contextual
- [ ] Responsive design optimizado para móviles
- [ ] Accessibility compliance (WCAG 2.1 AA)

Technical Implementation:

- [ ] Install and configure notification library (toastr/sweetalert)
- [ ] Implement loading spinners for AJAX operations

- [ ] Create user-friendly error message mapping
- [ ] Add breadcrumb navigation component
- [ ] Implement inline form validation
- [ ] Add progress indicators for multi-step processes
- [ ] Optimize mobile touch interactions
- [ ] Implement keyboard navigation support

UI Components to Create:

- [ ] Loading spinner component
- [ ] Toast notification system
- [ ] Breadcrumb navigation
- [ ] Form validation feedback
- [ ] Progress bars/indicators
- [ ] Modal dialogs for confirmations
- [ ] Tooltip help system

JavaScript Enhancements:

```javascript
// Form validation feedback
class FormValidator {
  static validateField(field) {
    const value = field.value.trim();
    const rules = this.getValidationRules(field);

    // Real-time validation with visual feedback
    if (this.isValid(value, rules)) {
      this.showValidState(field);
    } else {
      this.showInvalidState(field, this.getErrorMessage(rules));
    }
  }
}

// Progress tracking
class OperationProgress {
  static updateProgress(step, total, message) {
    const percentage = (step / total) * 100;
    document.getElementById('progress-bar').style.width = `${percentage}%`;
    document.getElementById('progress-message').textContent = message;
  }
}
```

Accessibility Improvements:

- [ ] ARIA labels for all interactive elements
- [ ] Keyboard navigation for all functionality
- [ ] Color contrast compliance (4.5:1 ratio minimum)
- [ ] Screen reader compatible
- [ ] Focus management for dynamic content
- [ ] Alternative text for images
- [ ] Semantic HTML structure

Mobile Optimizations:

- [ ] Touch-friendly button sizes (44px minimum)
- [ ] Swipe gestures where appropriate
- [ ] Optimized form inputs for mobile keyboards
- [ ] Improved tap targets spacing
- [ ] Fast click implementation

Performance UX:

- [ ] Perceived performance improvements
- [ ] Progressive loading of content
- [ ] Optimistic UI updates
- [ ] Background processing with feedback
- [ ] Image lazy loading
- [ ] CSS/JS minification and compression

Error Handling UX:

python

```python
# User-friendly error mapping
ERROR_MESSAGES = {
    'FileNotFoundError': 'No se pudo encontrar el archivo especificado. Verifique la ruta e intente nuevamente.',
    'PermissionError': 'No tiene permisos para acceder a este archivo. Contacte al administrador.',
    'ValueError': 'Los datos ingresados no son válidos. Revise el formato e intente nuevamente.',
    'ConnectionError': 'Problema de conexión. Verifique su conexión a internet e intente nuevamente.'
}
```

Testing Requirements:

- [ ] Cross-browser testing (Chrome, Firefox, Safari, Edge)
- [ ] Mobile device testing (iOS, Android)
- [ ] Accessibility testing with screen readers
- [ ] Performance testing (Lighthouse audit)

- [ ] Usability testing with real users

Analytics and Monitoring:

- [ ] User interaction tracking
- [ ] Error rate monitoring
- [ ] Page load performance tracking
- [ ] User flow analysis

Depends on: SENIAL-006

```
---

## 📊 MÉTRICAS DE SEGUIMIENTO

### Sprint Velocity Tracking:
```

Sprint 1 (EPIC-INFRA): 13 SP - Critical foundation Sprint 2 (EPIC-WEB): 11 SP - Framework modernization
Sprint 3 (EPIC-QUALITY): 18 SP - Code quality improvements Sprint 4 (EPIC-TEST): 21 SP - Testing and automation Sprint 5 (EPIC-DOC): 13 SP - Documentation and polish

Total: 76 Story Points across 5 sprints
Average: 15.2 SP per sprint

```
### Definition of Ready (DoR) - Before Sprint Planning:
- [ ] **Acceptance Criteria** clearly defined and measurable
- [ ] **Dependencies** identified and noted
- [ ] **Story Points** estimated by team
- [ ] **Technical approach** discussed and agreed upon
- [ ] **Risks and assumptions** documented
- [ ] **Testability** criteria defined

### Definition of Done (DoD) - Before Moving to Done:
- [ ] **Code Review** completed and approved
- [ ] **Unit Tests** written and passing (80%+ coverage)
- [ ] **Integration Tests** passing where applicable
- [ ] **Documentation** updated (README, inline docs)
- [ ] **Security Scan** passing (no critical vulnerabilities)
- [ ] **Performance** validated (no regression >10%)
- [ ] **Accessibility** verified for UI changes
- [ ] **Code Quality** checks passing (pylint >8.0)

---

## 🎯 PRIORIZACIÓN Y DEPENDENCIAS

### Critical Path:
```

SENIAL-001 (Python) → SENIAL-002 (Deps) → SENIAL-005 (Flask) → SENIAL-010 (Testing) → SENIAL-011 (CI/CD)

```

### Parallel Development Opportunities:
- **SENIAL-003** (Security) can run parallel to SENIAL-001
- **SENIAL-009** (Validation) independent after Sprint 1
- **SENIAL-012** (Docs) can start in Sprint 3
- **SENIAL-013** (UX) only depends on SENIAL-006

### Risk-Based Prioritization:
1. **🔴 Critical (Week 1):** SENIAL-001, SENIAL-003 (Security vulnerabilities)
2. **🟡 High (Week 2-4):** SENIAL-002, SENIAL-004, SENIAL-005 (Foundation)
3. **🟢 Medium (Week 5-8):** Quality and testing improvements
4. **🔵 Low (Week 9-10):** Documentation and UX polish

---

*Este backlog está listo para importación directa a Jira y proporciona una guía completa para la modernización del proyecto SenialSOLID.*
```