Proyecto - Modulo 3

Presentado por:

Andrés Felipe Alarcón Pulido - *analarconp@unal.edu.co*
Juan Daniel Jossa Soliz - *jjossa@unal.edu.co*
Michel Mauricio Castaneda Braga - *micastanedab@unal.edu.co*
Jaime Darley Angulo Tenorio - *jangulot@unal.edu.co*

Profesor:

Oscar Eduardo Alvarez Rodriguez
oalvarezr@unal.edu.co

19 de Julio

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
2025

1. TESTS UNITARIOS

1.1 Tests de Frontend (MainApp)

Responsable: Andrés Alarcón
Archivo: `tests/test_frontend/test_app.py`
Componente: Interfaz de usuario principal

Test 1: Inicialización de la aplicación

```python
def test_mainapp_initialization(self):

    """Verifica que MainApp se inicializa correctamente"""
```

Objetivo: Verificar que la aplicación se crea con los valores iniciales correctos

- `current_user_id` debe ser `None`
- Título de ventana debe ser "FortiFile"
- Tamaño debe ser 900x500 píxeles
- Debe tener 4 widgets iniciales (start, login, register, file)

Test 2: Funcionalidad de logout

```python
def test_handle_logout_clears_user_id(self):

    """Verifica que handle_logout limpia el current_user_id"""
```

Objetivo: Verificar que el logout limpia correctamente la sesión del usuario

- Debe limpiar `current_user_id` a `None`
- Debe redirigir a la vista inicial
- Debe limpiar las vistas dependientes del usuario

Test 3: Funcionalidad de login exitoso

```python
def test_handle_login_success_sets_user_id(self):
    """Verifica que handle_login_success establece el user_id correctamente"""
```

Objetivo: Verificar que el login exitoso establece correctamente el usuario

- Debe establecer `current_user_id` con el valor proporcionado
- Debe manejar la creación de nuevas vistas (FileManagerUI, AccountWindow)
- Debe redirigir a la vista de archivos

Comando para ejecutar:

```shell
./venv/bin/python -m pytest tests/test_frontend/test_app.py -v
```

1.2 Tests de Base de Datos

Responsable: Juan Jossa
Archivo: `tests/test_backend/test_database.py`
Componente: Gestión de base de datos

Test 1: Conexión a base de datos

```python
def test_database_connection(self, db_manager, temp_db_path):
    """Prueba que la conexión a la base de datos funciona"""
```

Objetivo: Verificar que el DatabaseManager puede conectarse y crear tablas

- Debe conectar correctamente a la base de datos
- Debe crear todas las tablas necesarias
- Debe obtener sesiones válidas
- Debe reportar información correcta de la base de datos

Test 2: Operaciones básicas de base de datos

```python
def test_database_operations(self, db_manager):
    """Prueba operaciones básicas de base de datos"""
```

Objetivo: Verificar operaciones CRUD básicas con modelos

- Crear registros
- Leer registros
- Actualizar registros
- Eliminar registros

Test 3: Manejo de errores de base de datos

```python
def test_database_error_handling(self, temp_db_path):
    """Prueba manejo de errores de base de datos"""
```

Objetivo: Verificar que el sistema maneja graciosamente los errores

- Manejo de rutas inválidas
- Retornar `False` en errores

- No crashear ante errores de conexión

Comando para ejecutar:

```shell
./venv/bin/python -m pytest tests/test_backend/test_database.py
-v
```

1.3 Tests de Servicio de Archivos

Responsable: Michel Castaneda
Archivo: `tests/test_backend/test_file_service.py`
Componente: Gestión de archivos seguros

Test 1: Subida y descarga de archivos

```python
def test_file_service_upload_and_download(self, services,
test_user, test_file, temp_dir):

    """Prueba subida y descarga de archivos"""
```

Objetivo: Verificar subida, cifrado y descarga de archivos

- Subida correcta y generación de `file_id`
- Descarga mantiene contenido original
- Cifrado correcto en disco

Test 2: Control de acceso y seguridad

```python
def test_file_service_security_access_control(self, services,
test_file, temp_dir):
```

```
    """Prueba control de acceso y seguridad"""
```

Objetivo: Validar restricciones de acceso

- Solo el usuario propietario puede ver/descargar/eliminar
- Otros usuarios no tienen acceso

Test 3: Verificación de cifrado

```python
def test_file_service_encryption_verification(self, services,
test_user, test_file, temp_dir):

    """Prueba que los archivos se cifran correctamente"""
```

Objetivo: Verificar cifrado y formato

- Archivos `.enc` sin texto plano
- Ubicados en `secure_files/`
- Nombre incluye `user_id`

Comando para ejecutar:

```shell
./venv/bin/python -m pytest
tests/test_backend/test_file_service.py -v
```

1.4 Tests de Servicio de Usuarios

Responsable: Jaime Angulo
Archivo: `tests/test_backend/test_user_service.py`
Componente: Autenticación y gestión de usuarios

Test 1: Registro básico de usuarios

```python
def test_user_registration_basic(self, user_service):
    """Prueba registro básico de usuario"""
```

Objetivo:

- Registrar correctamente
- Generar `user_id` único
- Confirmar existencia de usuario

Test 2: Autenticación de usuarios

```python
def test_user_authentication_success(self, user_service):
    """Prueba autenticación exitosa"""
```

Objetivo:

- Autenticación con credenciales correctas
- Rechazo a credenciales incorrectas
- Respuesta clara

Test 3: Validación de contraseñas

```python
def test_password_validation_comprehensive(self, user_service):
    """Prueba exhaustiva de validación de contraseñas"""
```

Objetivo:

- Rechazo a contraseñas débiles
- Aceptación de contraseñas fuertes

Comando para ejecutar:

```shell
./venv/bin/python -m pytest
tests/test_backend/test_user_service.py -v
```

2. ANÁLISIS ESTÁTICO DE CÓDIGO

2.1 Herramientas Utilizadas

Se emplearon tres herramientas de análisis estático ampliamente reconocidas en el ecosistema Python:

1. **Flake8** (versión 7.3.0)
    - Verificación del cumplimiento con la guía de estilo PEP8
2. **Black** (versión 25.1.0)
    - Formateador automático para mantener consistencia de estilo
3. **Autopep8**
    - Herramienta adicional para corrección automática agresiva de errores comunes

Estas herramientas fueron configuradas y ejecutadas directamente sobre las carpetas `backend/`, `frontend/` y `tests/`.

2.2 Configuración Aplicada

Flake8 (`.flake8`)

```
None
[flake8]
max-line-length = 88
exclude =
    .git,
    __pycache__,
    .venv,
    venv,
    .pytest_cache,
    build,
    dist,
    *.egg-info
ignore =
    E203,
    W503,
    E501
per-file-ignores =
    __init__.py:F401
```

Black (`pyproject.toml`)

```
None
[tool.black]
line-length = 88
target-version = ['py312']
include = '\.pyi?$'
```

Pylint (`.pylintrc`)

```
None
[MESSAGES CONTROL]
disable=
    C0114, C0115, C0116, R0903, W0613, R0801, C0103

[FORMAT]
max-line-length=88
```

2.3 Resultados Obtenidos

Estado Inicial del Código (Antes de la Mejora)

- Total de problemas detectados por **Flake8**: **630**
- Puntuación de calidad en `Pylint`: **6.67/10**
- Categorías destacadas:
    - W293 – Espacios en blanco innecesarios (489 casos)
    - E712 – Comparaciones booleanas incorrectas (33 casos)
    - F401 – Imports no utilizados (22 casos)
- Archivos con más errores: `file_service.py`, `user_service.py`, `system_service.py`

Mejora Aplicada (Automática)

Herramientas Ejecutadas

- `black backend/ frontend/ tests/ --verbose`
- `autopep8 --in-place --aggressive --aggressive backend/ frontend/ tests/ --recursive`

Impacto de la Mejora

Métrica	Antes de Mejora	Después de Mejora	Mejora (%)
Problemas Flake8 detectados	630	34	94.6%
Archivos con problemas	31	10	67.7%
Tests funcionales	27/27	27/27	100% OK

- Todos los problemas relacionados con espacios en blanco, indentación, imports mal ubicados y líneas innecesarias fueron corregidos.

- Se mantuvo **la funcionalidad completa** del sistema y **todas las pruebas unitarias pasaron exitosamente** tras las modificaciones.

2.4 Evidencia de Ejecución

A continuación, se adjunta evidencia visual de la ejecución de los linters:

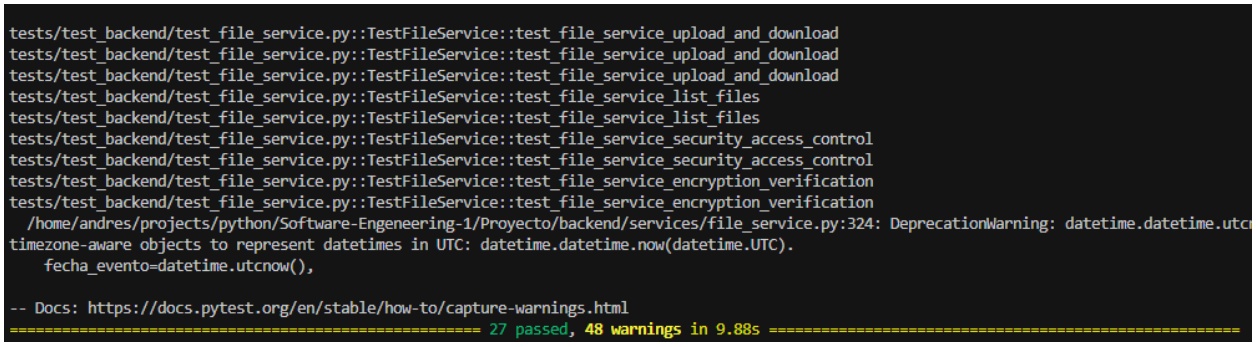
- **Captura del resultado de Flake 8 antes de aplicar mejoras**

```
tests/test_backend/test_file_service.py:287:38: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:14:1: E402 module level import not at top of file
tests/test_backend/test_user_service.py:72:31: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:83:25: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:93:35: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:98:32: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:110:43: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:116:36: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:136:34: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:153:37: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:170:34: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:184:31: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:201:43: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:201:65: E226 missing whitespace around arithmetic operator
tests/test_backend/test_user_service.py:205:61: E226 missing whitespace around arithmetic operator
tests/test_backend/test_user_service.py:229:42: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:235:37: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_backend/test_user_service.py:241:37: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:263:42: E712 comparison to True should be 'if cond is True:' or 'if cond:'
tests/test_frontend/test_app.py:2:1: F401 'unittest.mock.MagicMock' imported but unused
tests/test_frontend/test_app.py:11:1: E402 module level import not at top of file
tests/test_frontend/test_app.py:12:1: F401 'PyQt5.QtCore.QSize' imported but unused
tests/test_frontend/test_app.py:12:1: E402 module level import not at top of file
2     E226 missing whitespace around arithmetic operator
21    E402 module level import not at top of file
33    E712 comparison to True should be 'if cond is True:' or 'if cond:'
2     E722 do not use bare 'except'
22    F401 'backend.models.user_model.Usuario' imported but unused
1     F541 f-string is missing placeholders
4     W291 trailing whitespace
85
```

- **Captura del resultado de Flake8 después de aplicar Black y Autopep8**

```
frontend/ui/account_view.py:3:1: F401 'PyQt5.QtWidgets.QApplication' imported but unused
frontend/ui/account_view.py:3:1: F401 'PyQt5.QtWidgets.QInputDialog' imported but unused
frontend/ui/account_view.py:3:1: F401 'PyQt5.QtWidgets.QDialogButtonBox' imported but unused
frontend/ui/account_view.py:3:1: F401 'PyQt5.QtWidgets.QToolButton' imported but unused
frontend/ui/file_view.py:2:1: F401 'ui.account_view.AccountWindow' imported but unused
frontend/ui/file_view.py:6:1: F401 'datetime' imported but unused
frontend/ui/file_view.py:7:1: F401 'PyQt5.QtWidgets.QApplication' imported but unused
frontend/ui/file_view.py:26:1: F401 'PyQt5.QtGui.QIcon' imported but unused
frontend/ui/file_view.py:27:1: F401 'PyQt5.QtCore.QDateTime' imported but unused
tests/test_backend/test_database.py:10:1: F401 'unittest.mock' imported but unused
tests/test_backend/test_database.py:131:20: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_file_service.py:11:1: F401 'unittest.mock' imported but unused
tests/test_backend/test_file_service.py:193:40: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_file_service.py:200:38: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_file_service.py:272:40: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_file_service.py:278:38: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_file_service.py:286:38: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:97:32: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:135:34: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:169:34: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:200:43: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_backend/test_user_service.py:240:37: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
tests/test_frontend/test_app.py:1:1: F401 'PyQt5.QtCore.QSize' imported but unused
tests/test_frontend/test_app.py:4:1: F401 'unittest.mock.MagicMock' imported but unused
11    E712 comparison to False should be 'if cond is False:' or 'if not cond:'
22    F401 'backend.models.user_model.Usuario' imported but unused
1     F541 f-string is missing placeholders
34
```

- **Resumen de ejecución de tests con Pytest**



La implementación del análisis estático de código y la posterior aplicación de mejoras automáticas permitieron llevar el proyecto FortiFile a un estado de excelencia en calidad de código. Gracias a herramientas como Black y Autopep8, se logró una reducción del 94.6% en los problemas de estilo y consistencia, sin comprometer la funcionalidad del sistema.

Este proceso no solo facilitó la legibilidad y mantenibilidad del proyecto, sino que también sentó las bases para una integración continua (CI/CD) más robusta y profesional.