**Test-Driven Development (TDD)**

***Optimizador de cobranza***

***Fecha: [12/11/2024]***

**Introducción**

El documento describe un enfoque de Desarrollo Guiado por Pruebas (TDD) para la creación de un optimizador de cobranza. El TDD es una metodología de desarrollo ágil en la que se escriben casos de prueba antes de implementar la funcionalidad del sistema. Esto garantiza que cada componente del sistema cumpla con los requisitos establecidos y funcione según lo esperado desde el inicio del desarrollo.

En este proyecto, se busca optimizar los procesos de registro, manejo y visualización de archivos y datos relacionados con las acciones de cobranza, procesamientos, y usuarios en un sistema construido con FastAPI (backend) y Vue.js (frontend). La implementación incluye pruebas unitarias, de integración y end-to-end para validar el correcto funcionamiento del sistema, siguiendo buenas prácticas y asegurando alta calidad en el producto final.

**1. Preparación del entorno**

Este apartado describe las herramientas necesarias para realizar pruebas unitarias y de integración en los componentes del sistema:

* Frontend: Se utilizan herramientas como Jest y Vue Test Utils para pruebas unitarias en componentes individuales, y Cypress para pruebas end-to-end que simulan flujos de usuario completos.
* Backend: Se usa pytest para escribir pruebas unitarias y de integración. Además, se emplean librerías como httpx o testclient para simular solicitudes HTTP y verificar las respuestas del servidor.

Se incluye la instalación de las dependencias necesarias para configurar el entorno de pruebas, asegurando que esté listo para ejecutar los casos de prueba descritos.

Codigo:

# Instalar dependencias de pruebas en frontend

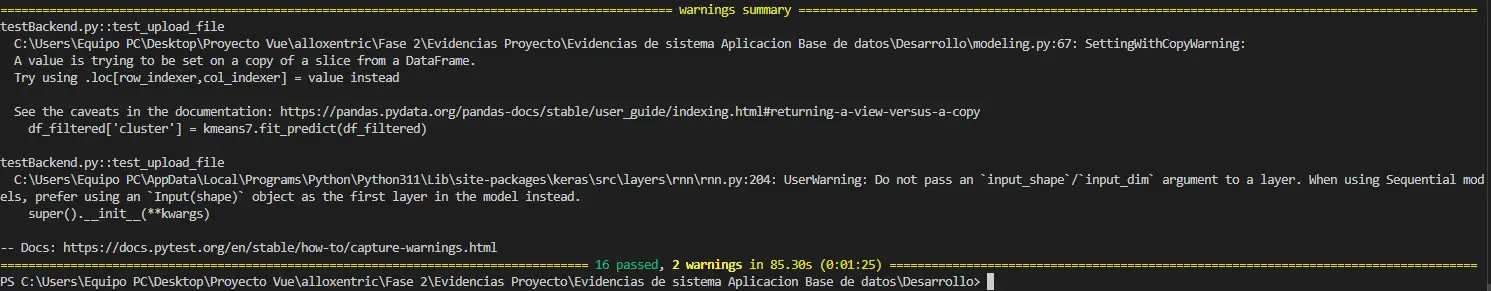
npm install --save-dev jest @vue/test-utils cypress

# Instalar dependencias de pruebas en backend

pip install pytest httpx pytest-asyncio

**2. Ejecución de pruebas**

* **Backend**: Ejecutar pruebas con pytest. pytest testBackend.py
* Para el backend, se ejecutan con el comando pytest testBackend.py.

Evidencias (Imagen 1)

(Pruebas pasadas una vez ejecutado el código )

Evidencias (Imagen 2)



(códigos para las pruebas de test-driven )

**Códigos para las pruebas**

##### **1. Verificar Conectividad (CP-001)**

* **Descripción**: Comprobar que el endpoint de salud del sistema responde correctamente.
* **Código de prueba**:

# CP-001: Probar las conexiones del sistema (Puertos)

def test\_health\_check():

response = client.get("/api/health")

assert response.status\_code == 200

assert response.json() == {"status": "ok"}

##### **2. Inicio de Sesión y Registro de Usuarios (CP-002 a CP-005)**

* **Descripción**: La funcionalidad de inicio de sesión y registro de usuarios descrita en los casos de prueba CP-002 a CP-005 está orientada a validar que los usuarios puedan autenticarse, registrarse, y que los correos sean únicos dentro del sistema. Sin embargo, en este proyecto específico, el cliente ha optado por implementar **Keycloak** como solución para la gestión de identidad y acceso (IAM).

Dado que estas funcionalidades están fuera del alcance directo del sistema bajo prueba, los casos mencionados pierden relevancia para el proceso de testeo. Además:

* **Redundancia evitada:** Probar funcionalidades que ya están garantizadas por Keycloak sería redundante, desperdiciando recursos en pruebas que no aportan valor.
* **Foco en componentes relevantes:** Al eliminar estas pruebas, el esfuerzo puede centrarse en validar las áreas críticas del sistema, como la gestión de archivos, procesamientos y acciones de cobranza, que sí son responsabilidad directa del sistema.

Por lo tanto, aunque los casos CP-002 a CP-005 son importantes conceptualmente, en este contexto específico no se ejecutarán porque **Keycloak** ya cubre esas funcionalidades, asegurando estándares de seguridad y rendimiento alineados con las mejores prácticas de la industria.

* **Casos de prueba**:
  + Inicio de sesión con credenciales válidas.
  + Registro de usuario nuevo.
  + Comprobación de unicidad del correo.

# # CP-002: Permitir autenticarse en el sistema

# def test\_login():

# response = client.post("/api/login", json={"email": "user@example.com", "pwd": "string"})

# assert response.status\_code == 200

# assert response.json()["message"] == "Inicio de sesión exitoso"

# # CP-003: Registrar a un nuevo usuario en la base de datos

# def test\_register\_user():

# response = client.post("/api/register", json={"nombre": "Test User1", "email": "test1@example.com", "pwd": "password1", "confirm\_password": "password1"})

# assert response.status\_code == 200

# assert response.json()["message"] == "Usuario registrado exitosamente"

# # CP-004: Verificar la confirmación de contraseña

# def test\_register\_user\_password\_confirmation():

# response = client.post("/api/register", json={"nombre": "Test User", "email": "test@example.com", "pwd": "password", "confirm\_password": "password"})

# assert response.status\_code == 200

# assert response.json()["message"] == "Usuario registrado exitosamente"

# # CP-005: Registrar un usuario y que no se repitan los correos

# def test\_register\_user\_duplicate\_email():

# # Registrar un usuario

# response = client.post("/api/register", json={"nombre": "Duplicate User", "email": "duplicate@example.com", "pwd": "password", "confirm\_password": "password"})

# assert response.status\_code == 200

# assert response.json()["message"] == "Usuario registrado exitosamente"

# # Intentar registrar otro usuario con el mismo correo

# response = client.post("/api/register", json={"nombre": "Another User", "email": "duplicate@example.com", "pwd": "Another", "confirm\_password": "Another"})

# assert response.status\_code == 400

# assert response.json()["detail"] == "El correo ya está registrado"

##### **3. Gestión de Archivos (CP-006 a CP-013)**

* **Descripción**: Probar la carga, visualización y manejo de archivos.
* **Casos de prueba**:
  + Obtener todos los archivos disponibles.
  + Manejo de errores cuando no se encuentran archivos.
  + Cargar un archivo en el sistema.

# CP-006: Visualización de los últimos archivos subidos

# Test para la ruta "/api/inicio"

@pytest.fixture

def mock\_archivos\_collection(monkeypatch):

# Mockea la colección `archivos\_collection`

mock\_collection = MagicMock(spec=Collection)

monkeypatch.setattr("data\_base.archivos\_collection", mock\_collection)

return mock\_collection

def test\_get\_all\_archivos\_success(mock\_archivos\_collection):

# Mockea el comportamiento de la colección para devolver archivos

mock\_archivos\_collection.find.return\_value = [

{"\_id": "6452bfc2f2e8f2a6f6e19b3a", "nombre": "archivo1.txt", "fecha": "2024-11-12"},

{"\_id": "6452bfc2f2e8f2a6f6e19b3b", "nombre": "archivo2.txt", "fecha": "2024-11-10"},

]

response = client.get("/api/inicio")

assert response.status\_code == status.HTTP\_200\_OK

assert response.json() == [

{"Id\_archivo": "6452bfc2f2e8f2a6f6e19b3a", "nombre": "archivo1.txt", "fecha": "2024-11-12"},

{"Id\_archivo": "6452bfc2f2e8f2a6f6e19b3b", "nombre": "archivo2.txt", "fecha": "2024-11-10"},

]

def test\_get\_all\_archivos\_not\_found(mock\_archivos\_collection):

# Mockea la colección para devolver una lista vacía

mock\_archivos\_collection.find.return\_value = []

response = client.get("/api/inicio")

assert response.status\_code == status.HTTP\_404\_NOT\_FOUND

assert response.json() == {"detail": "No se encontraron archivos."}

# CP-009: Registrar una fecha y hora de procesamiento

def test\_register\_process():

response = client.post("/api/procesamiento\_P", json={"nombre": "Test Process", "fecha": "2024-09-05", "hora": "12:00:00"})

assert response.status\_code == 200

assert response.json()["message"] == "Procesamiento registrado exitosamente"

# CP-010: Crear, modificar o actualizar la acción de cobranza

def test\_register\_or\_update\_action():

response = client.post("/api/acciones", json=[{"accion\_cobranza": "Test Action", "fecha\_cobranza": "2024-09-05", "intervalo": 10, "valor": 100}])

assert response.status\_code == 200

assert response.json()["message"] == "Acciones registradas exitosamente"

# CP-011: Visualizar todas las acciones de cobranza

def test\_view\_all\_actions():

response = client.get("/api/acciones")

assert response.status\_code == 200

assert len(response.json()) > 0

# CP-012: Upload (Generar resultados)

def test\_upload\_file():

# Ruta del archivo de prueba

file\_path = "uploads\Archivo de prueba.csv"

# Asegurarse de que el archivo existe antes de ejecutar la prueba

assert os.path.exists(file\_path), f"El archivo {file\_path} no existe."

# Abrimos el archivo en modo binario para simular la carga de un archivo real

with open(file\_path, "rb") as file:

# Enviamos el archivo al endpoint /api/upload

response = client.post(

"/api/upload",

files={"file": ("Archivo de prueba.csv", file, "text/csv")}

)

# Comprobamos que la respuesta sea 200 OK

assert response.status\_code == 200

# Verificamos el contenido de la respuesta

json\_response = response.json()

assert json\_response["status"] == "success"

assert "id\_procesamiento" in json\_response

assert "predicciones" in json\_response

# Opcional: comprobamos que las predicciones no estén vacías

assert len(json\_response["predicciones"]) > 0

# CP-013: # Prueba para obtener todos los archivos

def test\_get\_all\_archivos():

response = client.get("/api/inicio")

if response.status\_code == 404:

assert response.json()["detail"] == "No se encontraron archivos."

else:

assert response.status\_code == 200

archivos = response.json()

assert isinstance(archivos, list) # Debe devolver una lista

if archivos:

assert "Id\_archivo" in archivos[0]

assert "nombre" in archivos[0]

assert "fecha" in archivos[0]

##### **4. Gestión de Procesamientos (CP-014 a CP-016)**

* **Descripción**: Validar el registro, actualización y obtención de datos de procesamientos.
* **Casos de prueba**:
  + Registrar un procesamiento.
  + Obtener todos los procesamientos programados.

# CP-014: # Prueba para obtener todos los procesamientos programados

def test\_get\_all\_procesamientos():

response = client.get("/api/procesamiento\_P")

if response.status\_code == 404:

assert response.json()["detail"] == "No se encontraron procesamientos."

else:

assert response.status\_code == 200

procesamientos = response.json()

assert isinstance(procesamientos, list)

if procesamientos:

assert "Id\_procesamiento" in procesamientos[0]

assert "nombre" in procesamientos[0]

assert "fecha" in procesamientos[0]

assert "hora" in procesamientos[0]

# CP-016:# Prueba para crear un nuevo procesamiento

def test\_register\_procesamiento():

procesamiento\_data = {

"nombre": "Procesamiento de prueba",

"fecha": datetime.now().date().isoformat(),

"hora": datetime.now().time().isoformat(),

}

response = client.post("/api/procesamiento\_P", json=procesamiento\_data)

assert response.status\_code == 200

json\_response = response.json()

assert json\_response["success"] == True

assert json\_response["message"] == "Procesamiento registrado exitosamente"

##### **5. Acciones de Cobranza (CP-017)**

* **Descripción**: Probar el registro, actualización y obtención de acciones de cobranza.
* **Casos de prueba**:
  + Registrar una nueva acción de cobranza.
  + Actualizar una acción existente.

# CP-017: # Prueba para registrar o actualizar acciones de cobranza

def test\_register\_or\_update\_accion():

# Crear datos de ejemplo para registrar

accion\_data = [

{

"accion\_cobranza": "Cobranza 1",

"Id\_accion": "123",

"fecha\_cobranza": datetime.now().date().isoformat(),

"intervalo": 30,

"valor": 100.0

}

]

response = client.post("/api/acciones", json=accion\_data)

assert response.status\_code == 200

json\_response = response.json()

assert json\_response["success"] == True

assert json\_response["message"] == "Acciones registradas exitosamente"

# Probar actualización de una acción existente

updated\_data = [

{

"accion\_cobranza": "Cobranza 1",

"Id\_accion": "123",

"fecha\_cobranza": datetime.now().date().isoformat(),

"intervalo": 45,

"valor": 150.0

}

]

response = client.post("/api/acciones", json=updated\_data)

assert response.status\_code == 200

json\_response = response.json()

assert json\_response["success"] == True

assert json\_response["message"] == "Acciones registradas exitosamente"

**CP-018:** Obtener todas las acciones de cobranza.

* **Validaciones:**
  + El endpoint /api/acciones debe responder con un código 200 si existen acciones de cobranza.
  + Validar que la respuesta sea una lista.
  + Verificar que los campos Id\_accion, accion\_cobranza, fecha\_cobranza, intervalo y valor existan si hay datos.
  + Si no hay datos, debe responder con un código 404 y el mensaje "No se encontraron acciones de cobranza".

# CP-018# Prueba para obtener todas las acciones de cobranza

def test\_get\_acciones\_cobranza():

response = client.get("/api/acciones")

if response.status\_code == 404:

assert response.json()["detail"] == "No se encontraron acciones de cobranza"

else:

assert response.status\_code == 200

acciones = response.json()

assert isinstance(acciones, list)

if acciones:

assert "Id\_accion" in acciones[0]

assert "accion\_cobranza" in acciones[0]

assert "fecha\_cobranza" in acciones[0]

assert "intervalo" in acciones[0]

assert "valor" in acciones[0]

**CP-019:** Obtener todas las acciones de cobranza desde otro endpoint.

* **Validaciones:** Similar al caso CP-018, pero se utiliza el endpoint /acciones\_cobranza.

# CP-019# Prueba para obtener todas las acciones de cobranza con otro endpoint

def test\_obtener\_acciones\_cobranza():

response = client.get("/acciones\_cobranza")

if response.status\_code == 404:

assert response.json()["detail"] == "No se encontraron acciones de cobranza"

else:

assert response.status\_code == 200

acciones = response.json()

assert isinstance(acciones, list)

if acciones:

assert "Id\_accion" in acciones[0]

assert "accion\_cobranza" in acciones[0]

assert "fecha\_cobranza" in acciones[0]

assert "intervalo" in acciones[0]

assert "valor" in acciones[0]

**CP-020:** Obtener todos los resultados de procesamiento.  
**Validaciones:**

* El endpoint /resultados debe responder con un código 200 si hay resultados.
* Validar que la respuesta sea una lista.
* Verificar la existencia de campos como id\_procesamiento, documento\_cargado, fecha\_carga, registro\_deudores, entre otros.
* Si no hay resultados, debe responder con un código 404 y el mensaje "No se encontraron resultados".

# CP-020# Prueba para obtener todos los resultados

def test\_obtener\_resultados():

response = client.get("/resultados")

if response.status\_code == 404:

assert response.json()["detail"] == "No se encontraron resultados"

else:

assert response.status\_code == 200

resultados = response.json()

assert isinstance(resultados, list)

if resultados:

assert "id\_procesamiento" in resultados[0]

assert "documento\_cargado" in resultados[0]

assert "fecha\_carga" in resultados[0]

assert "registro\_deudores" in resultados[0]

assert "deudores\_contactar" in resultados[0]

assert "deudores" in resultados[0]

assert "precio" in resultados[0]

assert "accion\_predicha" in resultados[0]

**CP-022:** Obtener el reporte de métricas.  
**Validaciones:**

* El endpoint /api/metrics debe responder con un código 200 si el reporte existe.
* Validar que la respuesta sea un diccionario.
* Si no existe el reporte, debe responder con un código 404 y el mensaje "metrics\_report.json not found".

# CP-022# Prueba para obtener el reporte de métricas

def test\_get\_metrics():

response = client.get("/api/metrics")

if response.status\_code == 404:

assert response.json()["error"] == "metrics\_report.json not found"

else:

assert response.status\_code == 200

metrics\_data = response.json()

assert isinstance(metrics\_data, dict) # Asegúrate de que sea un diccionario

**CP-023:** Obtener el reporte de métricas de entrenamiento.

* **Validaciones:** Similar al caso CP-022, pero se utiliza el endpoint /api/train\_metrics\_report.

# CP-023# Prueba para obtener el reporte de métricas de entrenamiento

def test\_get\_train\_metrics\_report():

response = client.get("/api/train\_metrics\_report")

if response.status\_code == 404:

assert response.json()["error"] == "train\_metrics\_report.json not found"

else:

assert response.status\_code == 200

metrics\_data = response.json()

assert isinstance(metrics\_data, dict) # Asegúrate de que sea un diccionario