MANUAL DE CONFIGURACIÓN Y FUNCIONALIDAD RETO CHELITA SOFTWARE – FULLSTACK TEST

Ana Gabriela Ordoñez Güemes
Chelita Software
Documentación técnica
`Reto Fullstack Test`

25 de febrero de 2026

RESUMEN

Este documento describe la configuración del proyecto (backend en FastAPI y frontend en React), los requisitos del sistema y la explicación de la funcionalidad de la aplicación: creación de documentos PDF a partir de un formulario y descarga mediante un código único de 10 caracteres.

***P*alabras clave** FastAPI · React · PDF · API REST · Vite · ReportLab

1. INTRODUCCIÓN–

La aplicación permite:

- **Crear un documento PDF** rellenando un formulario con nombre, apellido, edad, teléfono y correo. El sistema genera un PDF con una plantilla fija y devuelve un **código único de 10 caracteres** que identifica ese documento.
- **Descargar un PDF ya creado** introduciendo el código de 10 caracteres en el frontend; la aplicación devuelve el documento en base64 y se descarga en el navegador.

La arquitectura consta de:

- **Backend**: API REST en Python con FastAPI que genera los PDF (ReportLab), almacena los documentos en memoria y expone dos endpoints.
- **Frontend**: aplicación React (Vite) con formulario para crear documentos y sección para descargar por código.

2. REQUISITOS DEL SISTEMA–

- **Python** 3.10 o superior (para el backend).
- **Node.js** 18 o superior y **npm** (para el frontend con Vite).
- Navegador web moderno (Chrome, Firefox, Edge, Safari).

3. CONFIGURACIÓN DE LIBRERÍAS (DEPENDENCIAS)–

Las dependencias del backend y del frontend (incluidas las de tests) se instalan por separado. Si la ruta del proyecto contiene espacios (por ejemplo `Chelita Software`), use comillas al escribir rutas completas o ejecute los comandos desde dentro de la carpeta correspondiente.

3.1. Librerías del backend–

El archivo `backend/requirements.txt` define todas las dependencias Python, tanto de ejecución como de tests:

- **Ejecución**: `fastapi`, `uvicorn[standard]`, `reportlab`, `pydantic`.
- **Tests**: `pytest`, `pytest-asyncio`, `httpx` (cliente para TestClient).

Instalación: desde la carpeta del proyecto, active el entorno virtual del backend e instale:

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

Si no existe `.venv`, créelo antes con `python3 -m venv .venv`. No use una ruta con espacios sin comillas (ej. `/ruta/Chelita Software/backend/.venv/bin/python"`).

3.2. Librerías del frontend–

El archivo `frontend/package.json` define dependencias de producción y de desarrollo (build y tests):

- **Producción**: `react`, `react-dom`.
- **Desarrollo/build**: `vite`, `@vitejs/plugin-react`.
- **Tests unitarios**: `vitest`, `@testing-library/react`, `@testing-library/jest-dom`, `@testing-library/user-event`, `jsdom`.
- **Tests E2E**: `cypress`, `start-server-and-test`.

Instalación: desde la carpeta del proyecto:

```
cd frontend
npm install
```

Con eso quedan instaladas todas las dependencias, incluidas las de tests. No es necesario instalar nada más para ejecutar `npm run test:run` ni `npm run e2e`.

4. CONFIGURACIÓN–

4.1. Configuración del backend (FastAPI)–

1. Navegar a la carpeta del backend:

```
cd backend
```

2. Crear y activar un entorno virtual:

```
python3 -m venv .venv
source .venv/bin/activate     # Linux/macOS
# En Windows: .venv\Scripts\activate
```

3. Instalar dependencias:

```
pip install -r requirements.txt
```

4. Iniciar el servidor:

```
uvicorn app.main:app --reload --port 8000
```

El backend quedará disponible en http://localhost:8000. La documentación interactiva (Swagger) está en http://localhost:8000/docs.

Parámetros opcionales:

- `-port`: puerto (por defecto 8000). Si se cambia, debe coincidir con la configuración del proxy del frontend o con la URL de la API en el frontend si se usa modo standalone.
- `-reload`: recarga automática al modificar código (solo desarrollo).

4.2. Configuración del frontend (React + Vite)–

1. Navegar a la carpeta del frontend:

```
cd frontend
```

2. Instalar dependencias:

```
npm install
```

3. Iniciar el servidor de desarrollo:

```
npm run dev
```

La aplicación se abrirá en http://localhost:5173. El proxy de Vite redirige las peticiones a `/create` y `/document` al backend en `http://localhost:8000`, por lo que **el backend debe estar en ejecución** para que el frontend funcione correctamente.

Archivo de configuración `vite.config.js`:

```
server: {
  port: 5173,
  proxy: {
    '/create': 'http://localhost:8000',
    '/document': 'http://localhost:8000',
  },
},
```

Si se cambia el puerto del backend, debe actualizarse la URL en `proxy`.

4.3. Alternativa sin Node/npm (HTML standalone)–

Si no se desea usar `npm install` ni `npm run dev`, puede utilizarse el archivo `frontend/index-standalone.html`, que carga React desde CDN y se conecta directamente al backend por URL.

1. Asegurarse de que el backend esté corriendo en `http://localhost:8000`.
2. Abrir en el navegador el archivo `frontend/index-standalone.html` (doble clic o "Abrir con" el navegador), o servir la carpeta `frontend` con un servidor HTTP y abrir ese archivo (por ejemplo, `python3 -m http.server 8080` y luego http://localhost:8080/index-standalone.html).

En este modo, la URL de la API está fijada en el HTML (`http://localhost:8000`). El backend debe tener CORS configurado para el origen desde el que se sirve la página (por ejemplo `null` para `file://` o el origen del servidor usado).

5. FUNCIONALIDAD–

5.1. Flujo general–

1. El usuario rellena el formulario "Crear documento PDF" con nombre, apellido, edad, teléfono y correo.
2. Al enviar el formulario, el frontend llama al endpoint `POST /create` del backend con esos datos.
3. El backend genera un PDF con la plantilla "Chelita Software - Fullstack Test" y los datos recibidos, lo guarda en memoria asociado a un código único de 10 caracteres alfanuméricos y devuelve ese código.
4. El frontend muestra el código y permite copiarlo al portapapeles.
5. Para descargar más tarde, el usuario introduce el código en la sección "Descargar PDF por código" y pulsa el botón. El frontend llama a `GET /document/{code}` y, con la respuesta en base64, provoca la descarga del PDF en el navegador.

5.2. API del backend–

5.2.1. POST /create–

Crea un nuevo documento PDF a partir de los datos enviados en el cuerpo de la petición.

Cuerpo de la petición (JSON):

```
{
  "nombre": "Alexis",
  "apellido": "Pina",
  "edad": "29",
  "telefono": "5581064181",
  "correo": "alexis.pina@chelita.com.mx"
}
```

Respuesta exitosa (200):

```
{
  "success": true,
  "document_code": "EJ21243DIK"
}
```

`document_code` es una cadena de exactamente 10 caracteres (letras mayúsculas y/o dígitos) que identifica de forma única el documento generado.

5.2.2. GET /document/{code}–

Devuelve el documento PDF asociado al `code` en formato base64.

Parámetros:

- `code`: cadena de exactamente 10 caracteres (el `document_code` devuelto por `POST /create`).

Respuesta exitosa (200):

```
{
  "success": true,
  "document_b64": "jwiou2he1287ehiuhwadkjhei27h2i7he1i2e="
}
```

`document_b64` es el contenido del PDF codificado en base64. El frontend decodifica este valor y genera un enlace de descarga (`data:application/pdf;base64,...`).

Errores:

- **400**: el código no tiene 10 caracteres.
- **404**: no existe ningún documento almacenado con ese código.

5.3. Contenido del PDF generado–

El PDF generado tiene la siguiente estructura:

- Título centrado: “Chelita Software - Fullstack Test”.
- A continuación, los campos y sus valores:
  - Nombre
  - Apellido
  - Edad
  - Telefono
  - Correo

Se genera con la librería ReportLab en el backend. Los documentos se almacenan en memoria (diccionario código → bytes del PDF); si se reinicia el servidor, se pierden.

5.4. Funcionalidad del frontend–

- **Formulario de creación**: campos obligatorios para nombre, apellido, edad, teléfono y correo; validación básica (required, tipo email). Al enviar, se muestra un indicador de carga y, al terminar, un mensaje de éxito o error y el código del documento.
- **Código del documento**: se muestra en un bloque destacado con botón "Copiar" para copiar el código al portapapeles.
- **Descarga por código**: campo de texto limitado a 10 caracteres (se fuerza mayúsculas), validación de longitud antes de enviar la petición y mensajes de error o éxito. Al recibir el PDF en base64, se dispara la descarga con nombre `documento-{code}.pdf`.
- **Accesibilidad**: etiquetas asociadas a los campos, mensajes de error con `role=.ªlert"`, estados de carga con `aria-busy` y enfoque visible para teclado.

6. EJECUCIÓN DE TESTS–

A continuación se indica cómo ejecutar cada suite de tests. Las dependencias necesarias se instalan con los pasos de la sección 3.

6.1. Resumen: comandos para ejecutar los tests–

Suite	Dónde ejecutar	Comando
Tests backend (pytest)	`backend/` con venv activado	`pytest -v`
Tests frontend (Vitest)	`frontend/`	`npm run test:run`
Tests E2E (Cypress)	`frontend/`	`npm run e2e`
Test E2E API (script)	`backend/` con venv, servidor en marcha	`python ../scripts/e2e_api_test.py`

6.2. Tests del backend (pytest)–

Desde la carpeta `backend`, con el entorno virtual activado (`source .venv/bin/activate` en Linux/macOS):

```
pytest -v
```

Se ejecutan tests unitarios del servicio de generación de PDF (longitud y unicidad del código, generación de bytes PDF), de los endpoints (creación, obtención por código, validación 422, errores 400/404 y health) y un test de integración del flujo completo (crear documento y obtenerlo por código).

6.3. Resultado de la ejecución (backend)–

A continuación se muestra la salida típica de una ejecución exitosa de `pytest -v` en el backend:

```
============================= test session starts =====
platform linux -- Python 3.13.3, pytest-8.3.4
rootdir: .../backend
configfile: pytest.ini
plugins: anyio, asyncio
collected 11 items

tests/test_api.py::test_create_document_... PASSED
tests/test_api.py::test_get_document_... PASSED
tests/test_api.py::test_get_document_invalid_404 PASSED
tests/test_api.py::test_get_document_bad_400 PASSED
tests/test_api.py::test_health PASSED
tests/test_api.py::test_create_validation_422 PASSED
tests/test_api.py::test_create_empty_field_422 PASSED
tests/test_integration_flow.py::test_full_flow... PASSED
tests/test_pdf_service.py::test_generate_code_length PASSED
tests/test_pdf_service.py::test_generate_code_... PASSED
tests/test_pdf_service.py::test_build_pdf_... PASSED

```

```
==================== 11 passed in 0.06s =================
```

6.4. Tests del frontend (Vitest)–

Desde la carpeta `frontend` (no es necesario tener el servidor `npm run dev` en marcha):

```
cd frontend
npm run test:run
```

Se ejecutan tests con Vitest y React Testing Library: cliente API (`createDocument`, `getDocument`, `downloadPdfFromBase64`) y componente App (formularios, validación, flujo crear documento y descargar por código).

6.5. Resultado de la ejecución (frontend)–

Salida típica de `npm run test:run`:

```
> vitest run

 RUN  v2.1.9

  src/api.test.js (6 tests) 9ms
  src/App.test.jsx (7 tests) 513ms

 Test Files  2 passed (2)
      Tests  13 passed (13)
   Duration  1.51s
```

6.6. Tests E2E con Cypress–

Desde la carpeta `frontend`. El comando `npm run e2e` arranca el servidor de desarrollo y luego ejecuta Cypress en modo headless; no hace falta tener el backend corriendo (la API se simula):

```
cd frontend
npm run e2e
```

Para ejecutar solo los tests E2E contra la API real (backend y frontend en marcha en sus puertos): `npm run e2e:live`. Para abrir la interfaz gráfica de Cypress: `npm run cypress:open`.

6.7. Test E2E de la API (script Python)–

Con el backend corriendo en http://localhost:8000, desde la carpeta `backend` con el entorno virtual activado:

```
cd backend
source .venv/bin/activate
python ../scripts/e2e_api_test.py
```

Salida esperada en caso de éxito:

```
E2E API: comprobando flujo crear -> obtener...
  Health OK
  Crear documento OK (codigo: XXXXXXXXXX)
  Obtener documento OK (PDF valido)
E2E API: todos los pasos OK.
```

7. RESUMEN DE ARCHIVOS RELEVANTES–

- `backend/requirements.txt`: dependencias Python (FastAPI, ReportLab, etc.).

- `backend/app/main.py`: aplicación FastAPI y definición de endpoints.
- `backend/app/pdf_service.py`: generación del código único y del PDF con ReportLab.
- `backend/app/store.py`: almacenamiento en memoria (código → PDF).
- `backend/app/models.py`: modelos Pydantic de request/response.
- `frontend/package.json`: dependencias y scripts npm (React, Vite).
- `frontend/vite.config.js`: configuración de Vite y proxy al backend.
- `frontend/src/App.jsx`: componente principal y flujo del formulario y descarga.
- `frontend/src/api.js`: llamadas a `/create` y `/document` y descarga desde base64.
- `frontend/src/components/`: componentes reutilizables (Card, FormField, Button, Message, CodeDisplay).

Manual elaborado por Ana Gabriela Ordoñez Güemes.