EJERCICIO EVALUABLE [DI + AADD + SGE]

MINI-ERP UTILIZANDO EL STACK 'FARM'

El objetivo del proyecto es el desarrollo de una aplicación web de gestión empresarial simplificada (ERP/CRM), diseñada para que pequeñas empresas puedan digitalizar su cartera de clientes y el control de facturación. El sistema se basa en una arquitectura moderna que utilice el stack FARM (FastAPI + React + MongoDB), separando la lógica de negocio y la persistencia de datos del frontend.

Visualización de clientes

Listado completo de todos los clientes registrados en el sistema

Gestión de Facturación

Visualización de facturas asociadas a cada cliente específico

Visualización de Datos

Cálculo automático de importes totales por cliente

Stack Tecnológico y Módulos del Proyecto

Tecnologías Principales

- **Frontend:** React (JavaScript)
- **Backend:** FastAPI (Python)
- **Base de datos:** MongoDB
- **Validación:** Pydantic
- **Comunicación:** HTTP + JSON

Módulos Integrados

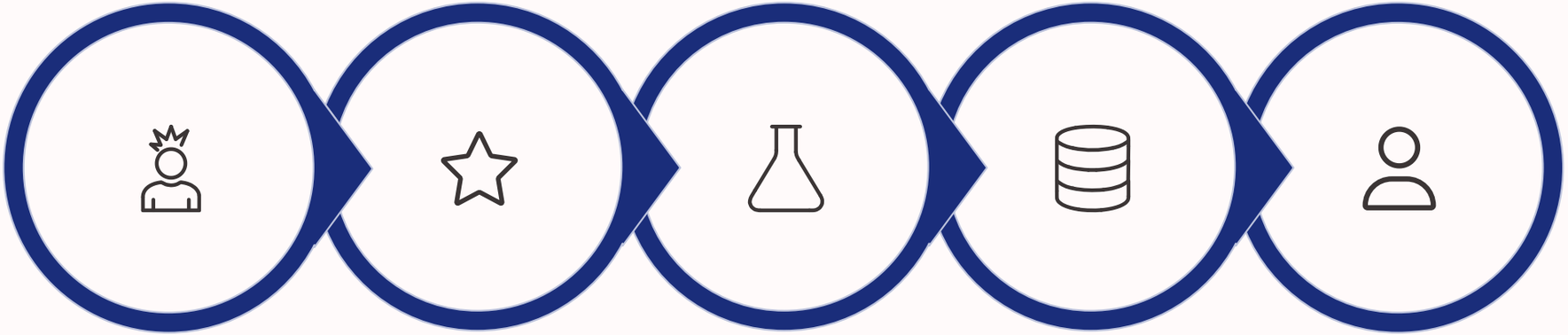
A. Módulo de Acceso a Datos: Implementación de base de datos NoSQL orientada a documentos (MongoDB) con modelado de colección de Clientes, conexión asíncrona y control de integridad de datos.

B. Módulo de Sistemas de Gestión: API RESTful con FastAPI que actúa como núcleo del sistema ERP para gestión de clientes y facturas.

C. Módulo de Desarrollo de Interfaces: Interfaz web dinámica y reactiva con React, componentización modular, consumo de API y gestión de estado.

Arquitectura General del Sistema

Esta arquitectura separa claramente cada responsabilidad del sistema, siguiendo las mejores prácticas de desarrollo moderno. El flujo de datos es unidireccional y bien definido, garantizando mantenibilidad y escalabilidad.



React **FastAPI** **models.py** **database.py** **MongoDB**

La comunicación entre capas se realiza exclusivamente mediante protocolo HTTP/JSON, asegurando una separación clara entre frontend y backend. Esta arquitectura permite que cada componente pueda ser desarrollado, probado y desplegado de forma independiente.

Capa de Presentación

React gestiona la interfaz de usuario y la experiencia interactiva

Capa de Lógica

FastAPI procesa peticiones y coordina operaciones

Capa de Datos

MongoDB proporciona persistencia flexible y escalable

Backend con FastAPI

main.py - Endpoints

El archivo main.py contiene los **endpoints**, que son los métodos públicos del sistema. Desde aquí se reciben las peticiones del frontend y se coordinan las operaciones del sistema.

```
from fastapi import FastAPI
from database import get_all_tasks

app = FastAPI()

@app.get("/tasks")
async def get_tasks():
    return await get_all_tasks()
```

1

GET

Consultar datos

2

POST

Crear datos

3

PUT

Modificar datos

4

DELETE

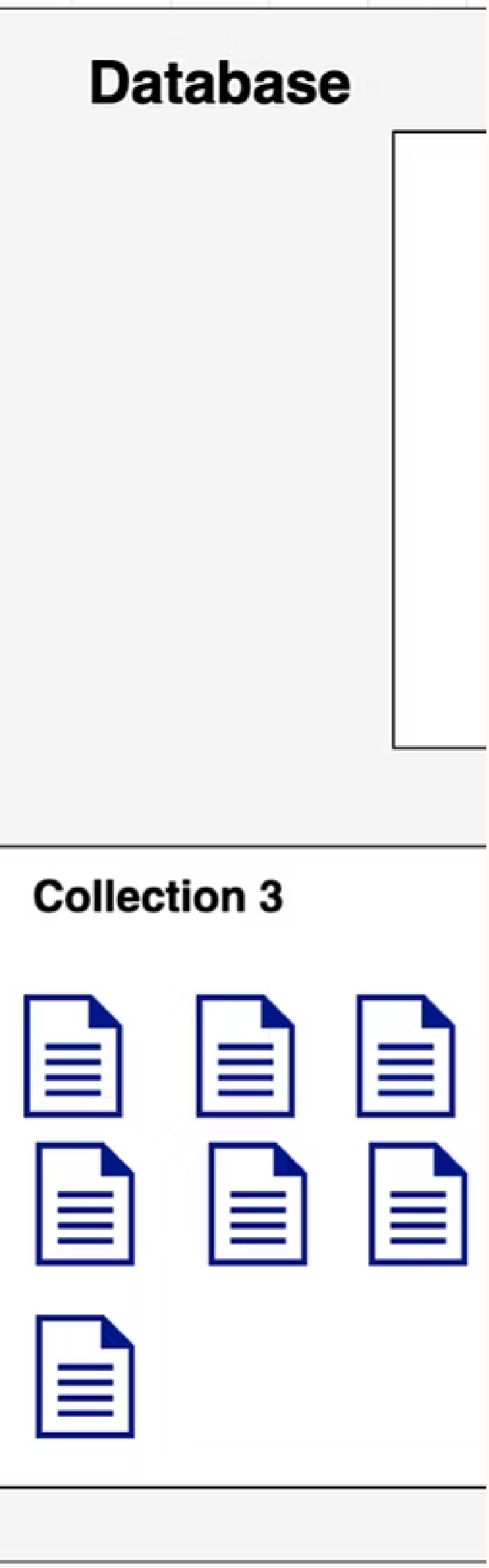
Eliminar datos

models.py - Modelos con Pydantic

Los modelos definen la estructura de los datos y validan la información automáticamente, garantizando la integridad de los datos en todo el sistema.

```
from pydantic import BaseModel
from typing import Optional

class Task(BaseModel):
 id: Optional[str] = None
 title: str
 description: Optional[str] = None
 completed: bool = False
```



Acceso a Datos y Persistencia

database.py - Operaciones CRUD

Este archivo se encarga de todas las operaciones con MongoDB. Aquí se implementa el CRUD completo. El resto de la aplicación **no accede directamente a la base de datos**, manteniendo una separación clara de responsabilidades.

```
from motor.motor_asyncio
import AsyncIOMotorClient

client =
AsyncIOMotorClient("mongodb:/
/localhost:27017")
db = client.todo_db
collection = db.tasks

async def get_all_tasks():
    tasks = []
    async for task in
collection.find():
        tasks.append(task)
    return tasks

async def insert_task(task):
    await
collection.insert_one(task)
```

MongoDB - Base de Datos

MongoDB almacena los datos de forma persistente. Cada tarea se guarda como un documento JSON flexible.

```
{
  "title": "Estudiar FastAPI",
  "description": "Ver el video del
proyecto",
  "completed": false
}
```

Crear

POST

Leer

GET

Actualizar

PUT

Eliminar

DELETE

Frontend con React y Sistema de Evaluación

Interfaz de Usuario

El frontend se encarga de mostrar formularios, tablas de datos y enviar peticiones al backend. La interfaz es responsive, intuitiva e incluye cabecera, zona de visualización y pie de página.

```
fetch("http://localhost:8000/tasks")
  .then(res => res.json())
  .then(data => console.log(data));
```

01

Componentización

Estructura modular del código con componentes reutilizables

02

Consumo de API

Peticiones al backend con gestión de cargas

03

Gestión de Estado

Manejo interno para reflejar cambios inmediatos

04

Diseño Responsive

Interfaz limpia, funcional y adaptable

Requisitos y Ampliación

Requisito Mínimo (8 puntos)

Sistema capaz de mostrar todos los datos de los clientes registrados en el sistema.

Ampliaciones (+1 punto cada una)

- Poder borrar clientes
- Poder actualizar datos de cliente
- Poder filtrar clientes
- Poder insertar nuevos clientes

Entrega, Evaluación y Conclusiones

1	2	3
Exposición de la APP Demostración del funcionamiento y explicación detallada del código implementado	**Repositorio de Código** Estructura clara en carpetas /backend y /frontend con documentación	**Memoria en PDF** Documento completo reflejando todo el trabajo realizado y código del proyecto

Conclusiones del Proyecto

Este proyecto demuestra la implementación exitosa de un sistema completo utilizando tecnologías modernas. Se ha logrado integrar conocimientos de tres áreas fundamentales: acceso a datos, sistemas de gestión empresarial y desarrollo de interfaces.

FastAPI con Endpoints API RESTful completa y funcional	**Modelos con Pydantic** Validación robusta de datos	**MongoDB** Persistencia flexible y escalable
Operaciones CRUD Sistema completo y funcional	**Arquitectura Clara** Diseño alineado con el temario	

Webgrafía: Inspirado en el video tutorial disponible en https://www.youtube.com/watch?v=7WE6v2EKm7M