🧾 INFORME DEL MÓDULO PROYECTOS

Proyecto: CODEASY — Sistema Gestor de Proyectos
Integrante responsable: Jose Alvarez
Entrega: Sprint 1 — Módulo de Proyectos
Tecnologías: Node.js, TypeScript, Fastify, PostgreSQL, Zod

🧠 1. Contexto general del sistema

El sistema **CODEASY** es una aplicación para la **gestión integral de proyectos, consultores y clientes**.
Permite administrar diferentes módulos mediante operaciones **CRUD** (Create, Read, Update, Delete) conectadas a una base de datos **PostgreSQL**, aplicando la **arquitectura hexagonal (Ports & Adapters)** para separar responsabilidades y mantener un diseño limpio y escalable.

Mi responsabilidad en este primer sprint fue el **módulo de Proyectos**, encargado de gestionar la información de los proyectos activos, garantizando la coherencia de datos y las validaciones temporales de fechas.

⚙️ 2. Objetivo del módulo de proyectos

El módulo **Proyectos** tiene como propósito permitir la **creación, lectura, actualización y eliminación lógica** de los registros de proyectos, asegurando integridad de la información, control de estado y validaciones coherentes sobre las fechas de inicio y entrega.

Funcionalidades implementadas:

1. **Creación de proyectos**
    - Inserta un nuevo registro en la base de datos.
    - Valida coherencia de fechas (inicio ≥ actual, entrega > inicio).
    - Asigna automáticamente `estado = "Creado"` y `estatus = "Activo"`.
2. **Lectura de proyectos**

- Lista todos los proyectos activos (`estatus = 'Activo'`).
- Permite obtener un proyecto por su `ID`.

3. **Actualización de proyectos**
   - Modifica campos como nombre, descripción, fechas o estado.
   - Valida que las fechas no sean anteriores a la actual y que la entrega sea posterior a la de inicio.
4. **Eliminación lógica**
   - Cambia el `estatus` a `"Eliminado"`, sin borrar físicamente el registro.
   - Permite conservar el historial de proyectos.
5. **Validaciones con Zod**
   - Estructura, tipos y formato correctos.
   - Validación de fechas con formato (`YYYY-MM-DD`).
   - Detección de campos no permitidos en los endpoints POST y PUT.

🧩 3. Arquitectura utilizada

El módulo se desarrolló bajo **arquitectura hexagonal**, separando las responsabilidades en capas bien definidas:

Capas del sistema:

- **Dominio (`core/dominio/proyecto`)** → define la entidad `Proyecto` y la interfaz `IProyecto`.
- **Aplicación (`core/aplicacion/casos-uso/Proyecto/ProyectoCasosUso`)** → implementa la lógica de negocio y validaciones.

- **Infraestructura (`core/infraestructura/postgres/ProyectoRepositorio.ts`)** → gestiona las operaciones SQL con PostgreSQL.
- **Presentación (`presentacion/`)** → contiene los controladores, rutas y validaciones Zod.
- **Common (`common/configuracion`)** → centraliza la configuración del entorno (.env).

💾 4. Base de datos

Se utilizó **PostgreSQL** con la siguiente tabla:

```
CREATE TABLE IF NOT EXISTS proyectos (

id SERIAL PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

descripcion VARCHAR(255) NOT NULL,

estado VARCHAR(20) DEFAULT 'Creado' CHECK (estado IN ('Creado', 'En proceso', 'Finalizado')),

estatus VARCHAR(20) DEFAULT 'Activo' CHECK (estatus IN ('Activo', 'Eliminado')),

id_cliente INT NOT NULL,

fecha_inicio DATE NOT NULL,

fecha_entrega DATE NOT NULL,

fecha_creacion TIMESTAMP DEFAULT NOW()

);
```

Reglas de negocio:

- `estado` inicial siempre es **"Creado"**.

- `estatus` inicial siempre es **"Activo"**.
- El método GET solo devuelve proyectos activos.
- Eliminación lógica mediante `estatus = 'Eliminado'`.

🧪 5. Validaciones implementadas (Zod + Casos de Uso)

Tipo de validación	Descripción	Ejemplo de mensaje
Estructural (Zod)	Tipos de datos, campos requeridos y formato.	`"La fecha de inicio debe tener un formato válido (YYYY-MM-DD)"`
Fechas de negocio (Casos de uso)	`fecha_inicio` ≥ actual, `fecha_entrega` ≥ actual y > `fecha_inicio`.	`"La fecha de entrega debe ser posterior a la fecha de inicio"`
Campos faltantes	Detección de campos requeridos ausentes.	`"Faltan campos requeridos: descripcion, id_cliente..."`
Campos no permitidos	Detección de propiedades extra.	`"Se enviaron campos no permitidos: campoExtra"`

🌐 6. Endpoints del módulo

Método	Ruta	Descripción
POST	`/api/proyecto`	Crea un nuevo proyecto.
GET	`/api/proyecto`	Lista todos los proyectos activos.
GET	`/api/proyecto/:idProyecto`	Obtiene un proyecto por su ID.
PUT	`/api/proyecto/:idProyecto`	Actualiza los datos de un proyecto.

PUT	`/api/proyecto/eliminar/:idProyecto`	Realiza la eliminación lógica.

⚙️ 7. Ejecución del módulo

Configuración del entorno

Archivo `.env`:

```
PUERTO=3000
PGHOST=localhost
PGPORT=5432
PGUSER=postgres
PGPASSWORD=admin
PGDBNAME=codeasy_db
```

Pasos para ejecución

1. Crear la base de datos y ejecutar la migración SQL.
2. Instalar dependencias con `npm install`.
3. Ejecutar el servidor con `npm run dev`.

Consola esperada:

```
Conectado correctamente a PostgreSQL
Servidor corriendo en http://localhost:3000
```

🧠 8. Pruebas realizadas

Prueba	Resultado esperado
Crear proyecto con todos los campos	✅ Proyecto creado correctamente
Crear con campo adicional	❌ “Se enviaron campos no permitidos”

Crear con fecha_inicio anterior a hoy	❌ “La fecha de inicio no puede ser anterior a la fecha actual”
Crear con fecha_entrega anterior a inicio	❌ “La fecha de entrega debe ser posterior a la fecha de inicio”
Actualizar solo descripción	✅ Proyecto actualizado correctamente
Actualizar con fecha inválida	❌ “Las fechas proporcionadas no son válidas”
Eliminar proyecto	✅ Estatus cambia a “Eliminado”
Listar proyectos	✅ Solo muestra los activos