PROYECTO BASE FULLSTACK

React + Express + MongoDB

Arquitectura Modular Reutilizable

1️⃣ FILOSOFÍA DEL PROYECTO

Este proyecto está diseñado para:

- Ser reutilizable
- Escalable
- Modular
- Separar responsabilidades
- Centralizar errores
- Replicar CRUDs fácilmente

El objetivo es que cualquier nuevo módulo siga exactamente la misma estructura.

2️⃣ ESTRUCTURA GENERAL DEL REPOSITORIO

/client → Frontend (React + Vite)
/server → Backend (Express + MongoDB)

Cada parte está completamente separada.

3️⃣ FRONTEND – ESTRUCTURA Y REGLAS

📂 Estructura

```
src/
├── api/
├── context/
├── features/
│   └── users/
│       ├── components/
│       ├── pages/
│       ├── services/
│       └── hooks/
├── styles/
```

🧩 Organización por Features

Cada módulo contiene:

- pages → vistas principales
- components → componentes del módulo
- services → llamadas a API
- hooks → lógica del módulo

Nunca mezclar responsabilidades.

🔁 Flujo Frontend

Formulario
→ Hook
→ Service
→ Axios
→ Backend
→ Respuesta
→ Hook
→ UI

Errores:

- Validation → se muestran en el formulario
- Server → alerta global (interceptor)

🎨 Sistema de Estilos

Carpeta:

styles/
 variables.css
 base.css
 components.css

Reglas:

- Clases reutilizables: `.input`, `.btn`, `.error-text`
- Variables CSS para colores y tamaños
- No estilos inline
- Consistencia visual en todos los módulos

4️⃣ BACKEND – ESTRUCTURA Y REGLAS

📂 Estructura

server/
├── models/
├── routes/
├── services/
├── validators/
├── middlewares/
├── utils/

🧠 Separación de responsabilidades

Carpeta	Responsabilidad
routes	Endpoints
validators	Validaciones de formato
services	Reglas de negocio
models	Esquemas DB
middlewares	Auth y roles
utils	Helpers

🔁 Flujo Backend

Request
→ authMiddleware
→ requireRole
→ validator
→ service
→ model
→ DB
→ respuesta

Errores → errorHandler global

5️⃣ MANEJO DE ERRORES

Tipos:

Validation

Campos específicos

```
{
 type: "validation",
 errors: { email: "Correo ya existe" }
}
```

Server

Error inesperado

```
{
 type: "server",
 message: "Error del servidor"
}
```

El middleware errorHandler centraliza todo.

---

6 CREAR UN NUEVO CRUD

En Backend

1. Crear modelo
2. Crear validator
3. Crear service
4. Crear route
5. Registrar en app.js

Seguir mismo patrón que users.

En Frontend

1. Crear carpeta en features
2. Crear service
3. Crear hook
4. Crear page
5. Crear componentes
6. Usar interceptor global

7️⃣ PRINCIPIOS CLAVE

- No lógica de negocio en routes
- No consultas directas desde componentes
- No duplicar validaciones
- No romper la estructura modular
- PATCH dinámico (solo actualiza campos enviados)

8️⃣ CUÁNDO MODIFICAR ALGO

Modificar:

- Validators → si cambia formato
- Services → si cambian reglas de negocio
- Styles → si cambian decisiones visuales globales
- Hooks → si cambia comportamiento UI

Nunca mezclar capas.

FIN DEL DOCUMENTO IMPRIMIBLE

Ahora te entrego los archivos listos para pegar:

📘 README.md

Proyecto Base Fullstack

React + Express + MongoDB

Arquitectura modular reutilizable para construir CRUDs de manera escalable y ordenada.

Instalación

Backend
cd server
npm install
npm run dev

Frontend
cd client
npm install
npm run dev

Estructura General

/client → Frontend
/server → Backend

Ver documentación detallada en:

- docs/ARCHITECTURE.md
- docs/FLOW.md

Características

- Arquitectura modular
- CRUD replicable
- Manejo global de errores
- Validaciones separadas por capa
- Sistema de estilos consistente
- Autenticación con roles

Filosofía

Separación clara de responsabilidades y reutilización de patrones.

Este proyecto puede usarse como base para nuevos sistemas administrativos o paneles internos.