Project Description — Modern Company Website

The Modern Company Website is a responsive, design-forward single-page application (SPA) built with React and Vite, created to showcase a design studio's portfolio, services, client relationships, and admin operations. The project emphasizes a polished UI, clear information architecture, and flexible connectivity options so it can run either as a standalone static site or as a full-stack application with a lightweight JSON backend or a MongoDB-powered production backend.

Frontend and user experience
The frontend uses React with modern tooling (Vite) for fast local development. Components are organized into pages—Home, Projects, Clients, Contact, and an Admin Panel—each tailored for specific user journeys: discovery (Home), portfolio browsing (Projects), client management (Clients), lead capture (Contact), and administrative control (AdminPanel). The UI is styled using utility-first classes and focuses on readable layouts, accessible form controls, and responsive components that adapt smoothly from mobile to large screens. Animations and micro-interactions are applied selectively via motion utilities to enhance perceived quality without impacting performance.

Backend options and architecture
To support both rapid prototyping and production readiness, the repository includes two backend approaches:

- Lightweight JSON backend: A small Node/Express server persists data to a local JSON file. This option is ideal for demos and development on machines where installing native modules is undesirable. It provides the essential REST endpoints for Projects, Clients, Contact messages, and Subscribers, enabling live CRUD operations without an external database.
- MongoDB backend (production-mode): A separate Express + Mongoose backend scaffold is provided for a full-stack deployment. This backend exposes the same logical endpoints but persists to MongoDB (e.g., Atlas), supports proper schema validation with Mongoose, and is ready for deployment to platforms like Render. Environment variables (MONGO_URI, PORT) configure connection details.

Flexible frontend API layer
The frontend includes an API abstraction that automatically chooses between remote and local modes. If VITE_API_URL is set, the app calls the remote backend; if not, the API layer falls back to localStorage-based mock persistence. This ensures the UI behaves consistently whether a backend is running or not, allowing developers to demo the UI instantly and later connect to a real backend without code changes.

Developer ergonomics and scripts

The root package contains convenience scripts to run the frontend and the lightweight backend concurrently for local development (npm run dev:all). The servers use node --watch for quick reloads during development; Node 18+ is recommended. The repository's README outlines clear run instructions for all three modes: frontend-only, JSON server, and Mongo backend. Important notes include the default ports, the required .env variables for the production backend, and how to point the frontend to a running API using VITE_API_URL.

Admin capabilities and UX features

The Admin Panel offers CRUD operations for projects and clients, a messages view for contact submissions, and a subscriber list for newsletters. Actions performed in the admin UI interact with the API layer; when the backend is remote, changes persist there, otherwise they are saved to localStorage. UI improvements include confirm-on-delete prompts, transient toast notifications for user feedback, and modal dialog flows for adding new items. These small UX details make the admin experience robust during testing and in real deployment.

Testing and verification

The project includes explicit guidance for testing APIs: GET endpoints can be checked via a browser; POST endpoints via Postman or Thunder Client; and frontend integration by inspecting browser DevTools (Network and Console). The API layer and server endpoints return clear JSON responses so testers can verify add/update/delete operations and confirm that data appears in the frontend.

Why this setup

This dual-mode architecture—UI-first with optional backend connectivity—caters to a range of use cases: quick portfolio demos, assignments requiring a full-stack submission, and staged deployment workflows. It reduces friction for designers and developers who want to present a working product quickly while enabling a straightforward upgrade path to a production-ready backend when required.

Next practical steps

To demo locally, run the lightweight server and frontend together with the provided npm scripts, or run the frontend alone for a pure static demo. For production, provision a MongoDB instance, set the MONGO_URI, and run the Mongo backend. For any help running or deploying the project, I can walk through terminal commands, fix environment issues, or prepare a deployment-ready configuration.