**A blue screen with white text

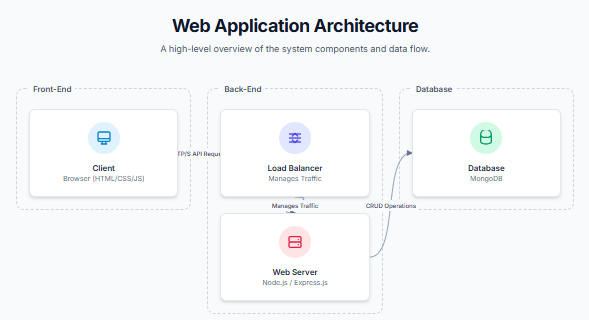
AI-generated content may be incorrect.**

**🧩 Architecture & Design Document**

**1. High-Level Architecture Diagram**

Our Project follows a standard 3-tier architecture:

**Overview**

**Description**

* **Frontend** is built with HTML/CSS/JavaScript. It provides a static interface with pages like index.html, dashboard.html, and components.html.
* **Backend** is powered by Node.js and Express.js. It handles API requests, authentication, and communication with the database.
* **Database** is MongoDB, integrated via Mongoose for schema modeling and data validation.

**2. Database Schema Diagram**

The database consists of three main collections: users, components, and movements, designed for flexibility and scalability using a NoSQL approach.

*A screenshot of a computer

AI-generated content may be incorrect.*

**🔐 User**

* name: String
* email: String (unique)
* password: String (hashed)
* role: String (user, admin)

**📦 Component**

* name: String
* description: String
* version: String
* createdAt: Date

**🔔 Notification**

* message: String
* type: String (info, warning, error)
* createdAt: Date

**3. API Endpoints**

**🔐 Authentication**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/auth/register | Register a new user |
| POST | /api/auth/login | Log in and get JWT token |

**📦 Components**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /api/components | Fetch all components |
| POST | /api/components | Add new component (Admin only) |
| DELETE | /api/components/:id | Delete component (Admin only) |

**🔔 Notifications**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /api/notifications | Fetch all notifications |
| POST | /api/notifications | Add new notification (Admin only) |

**4. Technology Justification**

| **Technology** | **Reason for Selection** |
| --- | --- |
| **Node.js** | Event-driven, non-blocking I/O model — ideal for building APIs |
| **Express.js** | Minimal and flexible web framework, great middleware support |
| **MongoDB** | NoSQL flexibility for evolving schemas and fast development |
| **Mongoose** | Schema modeling, built-in validation, and cleaner MongoDB queries |
| **JWT** | Stateless authentication — scalable and secure session management |
| **HTML/CSS/JS** | Easy to implement and host, suitable for a functional admin dashboard prototype |

**5. Scalability & Maintainability**

**💡 Scalability Considerations**

* **Stateless backend** allows horizontal scaling using container orchestration tools like Docker + Kubernetes.
* **MongoDB** supports replication and sharding, making it capable of handling high loads.
* **Environment-based configuration** enables easy deployment across development, staging, and production.

**🛠️ Maintainability Principles**

* Modular folder structure (models, middleware, routes) keeps logic clean and maintainable.
* Single-responsibility coding ensures each module does only one job (e.g., authentication, routing, DB interaction).
* Easily extendable — new models or features can be added with minimal refactoring.