Technical Design Document (TDD)

Project Name: [Insert Project Title]

Developer: [Your Name]

Date: [YYYY-MM-DD]

---

1. Architecture & Tech Stack

Frontend: React (ES6 syntax), Tailwind CSS Backend: Node.js, Express.js Database: MongoDB (Mongoose ODM) Version Control: Git / GitHub Development Environment: VS Code

---

2. Database Schema (MongoDB)

Define the collections, data types, and relationships before creating Mongoose models.

Model: User

- **_id: ObjectId**
- **firstName: String (Required)**
- **email: String (Required, Unique)**
- **passwordHash: String (Required)**
- **role: Enum ['admin', 'client'] (Default: 'client')**
- **createdAt: Timestamp**

Model: [Resource Name]

- **_id: ObjectId**
- **user: ObjectId (Ref: 'User')**
- **status: Enum ['pending', 'completed']**
- **details: Object**
- **createdAt: Timestamp**

---

3. API Endpoints (Express.js)

RESTful routing architecture mapping frontend requests to backend logic.

Method	Endpoint	Description	Auth Required	Body Payload
POST	/api/users/register	Creates a new user	No	{ email, password, firstName }
POST	/api/users/login	Authenticates and returns JWT	No	{ email, password }
GET	/api/[resource]	Fetches all records for a user	Yes (JWT)	None
POST	/api/[resource]	Creates a new record	Yes (JWT)	{ [relevant data] }
PUT	/api/[resource]/:id	Updates a specific record	Yes (JWT)	{ [fields to update] }
DELETE	/api/[resource]/:id	Deletes a record	Yes (Admin)	None

Export to Sheets

---

4. Backend Controllers & Functions

Modular breakdown of the /controllers directory to keep route files clean.

File: userController.js

- **registerUser(req, res): Hashes password, saves to DB, returns 201 status.**
- **loginUser(req, res): Compares passwords, generates JWT, returns 200 status.**
- **getUserProfile(req, res): Fetches user by ID from JWT payload.**

File: [resource]Controller.js

- **create[Resource](req, res): Validates input, saves to DB.**
- **get[Resource]s(req, res): Fetches DB records, applies filters/pagination.**
- **update[Resource](req, res): Finds by ID, updates fields, returns updated document.**

- **delete[Resource](req, res): Finds by ID, removes from DB.**

---

5. Frontend Architecture (React)

5.1. Page Routes

Top-level components rendered by React Router.

Route Path	Page Component	Description	Protected?
/	HomePage	The main landing page with marketing copy.	No
/login	LoginPage	Form for user authentication.	No
/dashboard	DashboardPage	Main hub displaying data summaries.	Yes
/[resource]/new	Create[Resource]Page	Form to create a new entry.	Yes

Export to Sheets

5.2. Component Breakdown

Modular pieces to build out the UI. Plan for React hooks (useRef, useMemo, useState) at the appropriate levels to avoid prop-drilling and unnecessary re-renders.

Global/Layout Components

- **Navbar: Top navigation and authentication toggles.**
- **Sidebar: Internal dashboard navigation.**
- **ProtectedRoute: Wrapper component checking for valid JWT.**

Dashboard Components

- **DashboardPage: Container managing server state and data fetching.**
- **DataSummaryWidget: Wrapper for statistics and recent activity.**
- **StatCard: Reusable UI for displaying numerical data.**
- **DataTable: Maps over database records to display rows.**

Form Components

- **InputField: Reusable text/email input with built-in Tailwind styling.**
- **SubmitButton: Reusable button handling loading states and onClick events.**

5.3. State Management Strategy

- **Global State: React Context API (e.g., AuthContext for user session and JWT).**
- **Server State: useEffect hooks for API fetching (or React Query).**
- **Local State: useState for specific component UI toggles (modals, mobile menus, form data).**

5.4. Styling Guidelines (Tailwind CSS)

- **Primary Colors: bg-blue-600, text-blue-900**
- **Secondary Colors: bg-gray-100, text-gray-600**
- **Alert/Error Colors: text-red-500, border-red-500**
- **Base Styling: Mobile-first responsive design using Tailwind's md: and lg: breakpoints.**

6. Third-Party APIs & Integrations

Define all external services, their purpose, and the required environment variables.

- **Google Maps API (Places/Geocoding)**
    - **Purpose:** Address autocomplete and validation on frontend forms.
    - **Implementation:** Loaded via script tag in index.html / React wrapper.
    - **Environment Variables:** REACT_APP_GOOGLE_MAPS_KEY (Frontend .env).
    - **Constraints:** Requires domain restriction in Google Cloud Console to prevent unauthorized use.
- **Google reCAPTCHA (v2/v3)**
    - **Purpose:** Spam and bot protection on submission forms.
    - **Implementation:** Frontend widget generates a token; Backend verifies the token via POST request to Google.
    - **Environment Variables:** REACT_APP_RECAPTCHA_SITE_KEY (Frontend), RECAPTCHA_SECRET_KEY (Backend).

- **[Insert API Name, e.g., Stripe, Twilio, SendGrid]**
    - **Purpose:** [What does it do?]
    - **Implementation:** [How is it integrated? e.g., Node SDK, REST fetch]
    - **Environment Variables:** [List required keys for .env]
    - **Constraints:** [e.g., Webhook setup required, Rate limits]