**DOCUMENTATION**

**1. Planned Approach and Architecture**

This project will utilize a **three-tier (or 3-layer) architecture** to separate concerns, making the application scalable and maintainable.

**Architecture Overview**

* **Presentation Layer (Frontend):** Built with **React** and responsible for the user interface, data input, visualization, and sending requests to the API.
* **Application Layer (Backend):** Built with **Node.js** and **Express.js**. This acts as the server, handling API requests, performing the core ROI calculations, and interacting with the database.
* **Data Layer (Database):** Uses **MySQL** to store application data, including user-defined scenarios and report requests.

**Data Flow**

1. User inputs scenario parameters into the **React** form.
2. React sends an API request (e.g., POST /api/simulate or POST /api/scenarios) to the **Express backend**.
3. The Express backend either: a. Calls the internal **ROI calculation logic**. b. Interacts with the **MySQL database** (using mysql2) to save, load, or list scenarios.
4. The backend sends the computed results or scenario data back to the **React frontend**.
5. React renders the results, including the key metrics and **Recharts** visualizations.

**2. Technologies, Frameworks, and Database**

| Category | Technology/Framework | Rationale |
| --- | --- | --- |
| **Frontend** | **React 18** | Modern, component-based library for building fast, interactive user interfaces. |
|  | **Tailwind CSS** | Utility-first CSS framework for rapid and responsive styling. |
| **Backend** | **Node.js** | High-performance, non-blocking runtime environment for the server. |
|  | **Express.js** | Minimalist, flexible Node.js web application framework for building APIs. |
|  | **mysql2** | Efficient, promise-based driver for connecting and querying MySQL from Node.js. |
| **Database** | **MySQL 8.0+** | Mature, robust relational database used to ensure data integrity and persistence of scenarios. |
| **Tools** | **dotenv** | Environment variable management for securely handling configuration (e.g., database credentials). |
|  | **Git** / **GitHub** | Version control and collaborative development. |

**3. Key Features and Functionality**

The application focuses on providing a full-featured tool for evaluating the financial benefits of automating accounts payable (AP) invoicing.

**Core Calculation & Display**

* **ROI Calculation Engine:** Implements the core financial logic to calculate monthly savings, payback period, and overall ROI based on user-defined inputs. It includes a **1.1x bias factor** to account for realistic automation efficiency improvements.
* **Real-Time Results:** Displays key metrics immediately upon calculation.
* **Data Visualization:** Uses interactive charts (**Recharts**) to show **cumulative savings** over the defined time horizon, providing a clear visual representation of the ROI.

**Data Persistence and Management (CRUD)**

* **Scenario Management:** Allows users to **Save** a set of inputs and calculated results as a named scenario to the MySQL database.
* **Load and Delete:** Provides full **CRUD** (Create, Read, Update, Delete) functionality to **Load** and **Delete** previously saved scenarios for comparison and re-analysis.
* **List Scenarios:** A dedicated API endpoint and UI feature to list all available scenarios.

**Reporting and User Experience**

* **Email Gating:** Implements an **email-gated report generation** feature. To receive the detailed report, the user must provide an email, which is captured in the report\_requests database table.
* **User-Friendly Interface:** A clear form layout built with **React** and **Tailwind CSS** ensures an intuitive and responsive experience across devices.
* **API Endpoints:** A well-structured backend API provides dedicated endpoints for all core functions: /simulate (calculate), /scenarios (save/list), and /report/generate.