Technical Documentation - Basic ERP Project

📑 1. Team Formation & Idea Development

Team Overview:

- Solo developer: Rayane Allaoui
- Roles: Project Manager, Lead Developer (Frontend & Backend), QA Engineer, Technical Writer

Brainstormed Ideas:

1. Task Manager
2. Personal Finance Tracker
3. Inventory Management System (ERP)

Evaluation Criteria:

- Feasibility
- Business value
- Scalability
- Reusability

Final MVP: Basic ERP System

- **Target Audience**: Small businesses
- **Problem Solved**: Centralized management of clients, products, and invoices
- **Core Modules**:
    - Authentication

    - Client Management
    - Product Catalog
    - Invoicing & Payment Tracking
- **Challenges Anticipated**:
    - Complex relationships in DB
    - UI/UX simplicity
    - Role-based access control

⥀ 2. Project Charter Development

Objectives:

- Simplify business operations for small companies.
- Provide a full-stack, modular ERP MVP.
- Ensure scalable, secure, and reusable codebase.

SMART Objectives:

1. Develop MVP ERP with 4 core modules in 6 weeks.
2. Ensure <1s response time for all API endpoints.
3. Achieve 90%+ test coverage for backend logic.

Stakeholders:

- Internal: Rayane Allaoui (Fullstack Dev)
- External: Mentors, Potential Users (Freelancers, SMBs)

Roles:

- Rayane: Project Management, DevOps, Fullstack Dev, QA, Documentation

Scope:

- **In-Scope**:
    - Web app (React frontend + Express backend)
    - Auth + Client + Product + Invoice modules
    - REST API
- **Out-of-Scope**:
    - Mobile app
    - Payroll or HR modules

Risks & Mitigations:

Risk	Mitigation
Delays due to solo dev	Use Kanban board and prioritize features
Security flaws	Use JWT + bcrypt + Helmet + HTTPS
UX complexity	Create simple wireframes first

High-Level Timeline:

- Week 1-2: Planning & Charter
- Week 3-4: Tech Docs + ERD + Mockups
- Week 5-6: MVP Development

- Week 7: QA & Testing
- Week 8: Deployment + Demo

⚙️ 3. Technical Documentation

1. User Stories & Mockups

- As a user, I want to create clients so I can issue invoices.
- As an admin, I want to manage products to keep catalog up to date.
- As a user, I want to view all invoices and their statuses.
- As a user, I want to login/logout securely.

Mockups: Home, Login, Dashboard, Client/Product/Invoice CRUD pages.

2. System Architecture

Components:

- Frontend: React.js + Axios
- Backend: Node.js + Express
- Database: MySQL
- Auth: JWT + bcrypt
- Framework: Vite
- Code organisation: back directory + front directory

```
[React] <--> [Express API] <--> [MySQL]

               |

            [JWT Auth]
```

3. DB Design (ER Diagram)

- `users(id, email, password_hash, role)`
- `clients(id, name, email, phone)`
- `products(id, name, price, stock)`
- `invoices(id, client_id, date, total, status)`
- `invoice_items(id, invoice_id, product_id, quantity)`

4. Sequence Diagrams

Login: User > React > Express (/login) > DB validation > JWT returned > React stores token

Create Invoice: User > React > Express (/invoices POST) > Insert invoice + items > Return success

5. API Documentation

Auth:

- `POST /auth/login`: `{ email, password }` → `{ token }`
- `POST /auth/register`: `{ email, password }`

Clients:

- `GET /clients` → list
- `POST /clients` → create

Products:

- `GET /products`
- `POST /products`

Invoices:

- `GET /invoices`
- `POST /invoices`
- `GET /invoices/:id`

6. SCM & QA Strategy

- GitHub repo: feature branches → pull requests → code review
- QA:
    - Unit tests: Jest (backend)
    - Manual tests: Postman for endpoints
    - Linter: ESLint + Prettier

7. Technical Choices

- **Express**: Simple, fast, middleware-friendly
- **MySQL**: Structured data, foreign keys
- **React**: Reusable components, SPA
- **JWT**: Secure and scalable token-based auth
- **Modular structure**: Scalable & maintainable codebase

END OF DOCUMENT