Home Owners Association Management System

1. Overview

The Home Owners Association Management System is a comprehensive web application designed to manage various aspects of property and association operations. It includes features for property management, budgeting, reporting, and document handling. The system is built using React.js for the frontend, Node.js for the backend, and SQLite for the database. It also includes scripts for generating PDFs, barcodes, and digital signatures.

2. Functional Requirements

2.1 Login Page

- User Authentication: Basic login functionality with default admin credentials.

- Security: Ensure secure authentication and authorization.

2.2 Home Page

- Property Management: Create, edit, and delete properties.

- Navigation: Access to various modules via buttons (Property Details, Unit Types, Owners List, Committee Details, Budgets, Reports, Documents, Settings).

2.3 Property Details

- Property Information: Store and manage property details (name, address, ownership title, association name, map location, logo, currency).

- Barcode and Stamp Generation: Convert property name into a barcode and stamp.

- PDF Generation: Print property details in an elegant PDF format.

2.4 Unit Types

- Unit Management: Create and manage different unit types (e.g., big apartment, small apartment, garage) with monthly fees.

- PDF Generation: Print unit details in an elegant PDF format.

2.5 Owners List

- Owner Information: Store owner details (full name, renter, address, phone, email, bank account number, unit type, ownership title number).
- CRUD Operations: Create, read, update, delete, and duplicate owner records.
- PDF Generation: Print owner details in an elegant PDF format.

2.6 Committee Details

- Committee Management: Create committee details using owner information.
- Position Assignment: Assign predefined or custom positions to committee members.
- Signature Generation: Convert President's name into a digital signature.
- PDF Generation: Print committee details in an elegant PDF format.

2.7 Regular Budget

- Yearly Budget Management: Manage yearly budgets starting from 2025.
- Income and Outcome: Track income and outcome for each month.
- Balance Calculation: Calculate and display the balance.
- PDF and XLS Generation: Generate PDF and XLS reports for income, outcome, and balance statements.
- In-App Notifications: Notify users about unpaid owners.

2.7.1 Income Matrix Table

- Columns: Months of the year (January to December), Deficit column, Sum column.
- Rows: Names of owners, Deficit row, Sum row.
- Cells:
  - Not Arrived Yet: White (for future months).
  - Paid: Green (with the amount and currency).

  - Not Paid: Red (with the amount preceded by a minus symbol and currency).
  - Partially Paid: Diagonally divided, green for paid amount, red for unpaid amount.
  - Paid in Advance: Blue.
  - Late Payment: Orange (with an increase option for late payment).

2.7.2 Outcome Matrix Table

- Columns: Months of the year (January to December), Sum column.
- Rows: Names of expenses (10 predefined, editable, removable, add new ones), Sum row.
- Cells:
  - Amount Entry: Enter the expense amount.
  - Justification: Upload a photo (optional).

2.8 Exceptional Budget

- Exceptional Budget Management: Create and manage exceptional budgets.
- Income and Outcome: Track income and outcome for exceptional budgets.
- PDF and XLS Generation: Generate PDF and XLS reports for exceptional budget statements.

2.9 Reports

- Report Aggregation: Group and organize all reports for easy access.
- PDF Generation: Generate PDF reports for various modules.

2.10 Documents

- Document Management: Upload, organize, and access documents.
- Search and Filter: Filter documents by date, time, size, and name.

2.11 Settings

- Backup and Restore: Backup and restore data.
- Profile Management: Manage user profiles.
- Language Selection: Switch between French and English.

3. Technical Requirements

3.1 Frontend

- Framework: React.js
- UI Components: Elegant and user-friendly interface.
- State Management: Use Redux or Context API for state management.
- Routing: React Router for navigation.

3.2 Backend

- Framework: Node.js with Express.js
- Database: SQLite for simplicity and ease of use.
- APIs: RESTful APIs for communication between frontend and backend.
- Authentication: JWT for secure authentication.

3.3 Database

- Schema Design: Design tables for properties, units, owners, committees, budgets, and documents.
- Data Integrity: Ensure data integrity and relationships between tables.

3.4 Scripts

- Barcode Generation: Use a library to generate barcodes.
- PDF Generation: Use a library like `pdfkit` to generate PDFs.
- Digital Signature: Use a library to generate digital signatures.

3.5 Deployment

- Start Scripts: `start.bat` and `start.sh` files to run the application locally.

- Environment Setup: Ensure the application runs smoothly on different operating systems.