Proto - Labour Management System Documentation Suite

Table of Contents

Project Overview: Construction Labour Management System

1. Introduction

This project is a comprehensive **Labour Management System** designed for the construction industry. It streamlines the management of daily wage labourers across multiple job sites, ensuring accurate tracking of attendance, payments, and work history.

Core Purpose

The primary goal is to digitize the manual process of tracking labour attendance and payments. It replaces paper-based muster rolls with a secure, digital system that provides real-time insights to administrators while empowering site supervisors with easy-to-use tools.

Target Audience

- **Administrators (Company Owners/Managers)**: Require high-level oversight of all sites, financial reports (salary, advances), and labour performance.
- **Site Supervisors**: Require a simple interface to mark daily attendance, record overtime, and issue advances for labourers at their specific site.

Key Problem Solved

- **Eliminates Proxy Attendance**: Secure login and locked daily submissions prevent tampering.
- **Accurate Wage Calculation**: Automates salary computation based on daily rates, attendance days, overtime, and deductions (advances).
- **Transparency**: Real-time data visibility for admins prevents discrepancies between site records and payroll.

---

2. Developer Setup Guide

Follow these steps to set up the development environment locally.

Prerequisites

- **Node.js**: v18.0.0 or higher.
- **npm**: v9.0.0 or higher.
- **Git**: Latest version.
- **Expo Go** app (on mobile device) or Android Studio/Xcode (for simulation).

Installation

1. **Clone the Repository**

   ```
   git clone <repository_url>
   cd Proto
   ```

2. **Server Setup**
   Navigate to the server directory and install dependencies.

   ```
   cd server
   npm install
   ```

   - **Environment Variables**: Create a `.env` file in `server/`.

     ```
     PORT=5000
     JWT_SECRET=your_super_secret_key_here
     ```

   - **Database**: The SQLite database (`proto.db`) will be automatically initialized when the server starts.

3. **Client Setup**
   Navigate to the client directory and install dependencies.

   ```
   cd ../client
   npm install
   ```

   - **API Configuration**: Open `client/src/constants.ts` and set `API_URL` to your machine's local IP address (e.g., `http://192.168.1.5:5000/api`). Do **not** use `localhost` if testing on a physical device.

Running the Application

1. **Start the Server**

   ```
   cd server
   node index.js
   # OR for development with auto-restart
   npx nodemon index.js
   ```

   Output should confirm: `Server running on http://localhost:5000`

2. **Start the Client**

```
cd client
npx expo start
```

   - Press `a` for Android Emulator.
   - Press `w` for Web.
   - Scan the QR code with Expo Go on your phone.

Linting & Code Quality

- **Linting**: Run `npm run lint` in the `client` directory to check for code style issues using ESLint.

---

3. Verification Scripts

The `server` directory contains several verification scripts to test core logic without the frontend:

- `node verify_attendance_lock.js`: Tests the daily locking mechanism.
- `node verify_overtime.js`: Tests overtime calculations.
- `node verify_labours.js`: Checks labour data integrity.

Architecture & Technology Stack

1. Technology Stack

This application follows a **Client-Server Architecture** using modern JavaScript frameworks.

Frontend (Client)

- **Framework**: React Native (https://reactnative.dev/) via Expo SDK 50 (https://expo.dev/)
- **Routing**: Expo Router v3 (https://docs.expo.dev/router/introduction/) (File-based routing)
- **Language**: TypeScript (`.ts`, `.tsx`)
- **State Management**: React Context & Hooks (`useState`, `useEffect`)
- **Networking**: Axios (HTTP Client)
- **UI/Design**: Styled components (React Native StyleSheet), `@expo/vector-icons`
- **Animations**: `react-native-reanimated`
- **Local Storage**: `@react-native-async-storage/async-storage` (for JWT & User Data)

Backend (Server)

- **Runtime**: Node.js (https://nodejs.org/)
- **Framework**: Express.js (https://expressjs.com/)
- **Database**: SQLite (https://www.sqlite.org/) (File-based relational DB)
- **ORM/Driver**: `sqlite` & `sqlite3` (Async/Await wrapper)
- **Authentication**:
   - `jsonwebtoken` (JWT Access & Refresh Strategy)
   - `bcryptjs` (Password Hashing)
- **Environment**: `dotenv` for configuration

---

2. System Architecture Diagram

```
graph TD
    User((User)) -->|Interacts| Client[Mobile Client (React Native)]

    subgraph Client App
        Client -->|Navigation| Router[Expo Router]
        Router -->|Render| Screens[Screens (Home, Attendance, Reports)]
        Screens -->|API Call| Services[Axios Service Layer]
    end

    Services -->|HTTP Request (JSON)| Server[Node.js / Express Server]

    subgraph Backend Server
        Server -->|Route Request| Routes[API Routes]
        Routes -->|Validate| Middleware[Auth Middleware (JWT)]
        Middleware -->|Controller Logic| Controllers[Business Logic]
        Controllers -->|SQL Query| DB_Driver[SQLite Driver]
    end

    DB_Driver -->|Read/Write| Database[(SQLite Database file)]

    Database -.->|Data| DB_Driver
    DB_Driver -.->|Result| Controllers
    Controllers -.->|JSON Response| Services
    Services -.->|Update State| Screens
```

---

3. Folder Structure Breakdown

Client Structure (`/client`)

- **`src/app`**: The core application logic and routing.
    - **`(tabs)`**: Contains the main tab navigator (Home, Sites, Reports).
    - **`(screens)`**: Standalone screens not in the tab bar (Auth, Profile).
    - **`_layout.tsx`**: Defins the root layout/navigation stack.
- **`src/components`**: Reusable UI components (Buttons, Cards, Modals).
- **`src/services`**: API integration.
    - `api.ts`: Central Axios instance with interceptors for token refresh.
- **`src/constants.ts`**: App-wide constants (API URL, Colors).

Server Structure (`/server`)

- **`index.js`**: Entry point. Initializes Express, Middleware, and Database.
- **`database.js`**: Handles SQLite connection and schema migrations/initialization.
- **`middleware/`**:
    - `auth.js`: Verifies JWT tokens and attaches user user info to `req.user`.
- **`routes/`**: Defines API endpoints.
    - `auth.js`: Signin/Signup logic.
    - `attendance.js`: Marking and viewing attendance.
    - `labours.js`: CRUD for labour profiles.
    - `sites.js`: Site management.
    - `overtime.js`: Overtime recording.
    - `reports.js`: Aggregated data for admin views.

---

4. Data Flow Patterns

Request Lifecycle

1. **User Action**: User taps "Submit Attendance" on the mobile app.
2. **Client Validation**: React State ensures all fields are filled.
3. **API Call**: `axios.post('/attendance')` is triggered.
    - The `Authorization` header (`Bearer <token>`) is attached automatically via interceptor.
4. **Server Authentication**: `authenticateToken` middleware verifies the JWT.
    - If valid, `req.user` is populated.
    - If expired, the client attempts to use the Refresh Token to get a new Access Token.
5. **Business Logic**:
    - The route handler verifies rules (e.g., "Is date in future?", "Is attendance locked?").
    - A Database Transaction (`BEGIN TRANSACTION`) starts.
6. **Database Operation**:
    - Multiple rows are inserted into `attendance`.
    - The `daily_site_attendance_status` table is updated to `locked = 1`.
    - `COMMIT` is executed if all succeed; `ROLLBACK` on error.
7. **Response**: Server sends `200 OK` JSON back to client.
8. **UI Update**: Client shows "Success" toast and navigates back.

API & Database Schema

1. Database Schema (SQLite)

Users (`users`)

Stores credentials for Admins and Supervisors.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`name`	TEXT	Full name
`phone`	TEXT	Mobile number (Login ID)
`password_hash`	TEXT	Bcrypt hashed password
`role`	TEXT	`admin` or `supervisor`
`created_at`	DATETIME	Timestamp

Labours (`labours`)

Profiles of all workers in the system.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`name`	TEXT	Full name

`phone`	TEXT	Contact number (optional)
`aadhaar`	TEXT	Identification number (optional)
`site_id`	INTEGER FK	Current assigned site (`sites.id`)
`rate`	REAL	Daily wage rate
`trade`	TEXT	Skill (e.g., Mason, Helper)
`status`	TEXT	`active`, `inactive`

Sites (`sites`)

Construction project locations.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`name`	TEXT	Project name
`address`	TEXT	Location details
`created_by`	INTEGER FK	Admin user ID

Attendance (`attendance`)

Daily attendance records per labourer.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`labour_id`	INTEGER FK	Worker ID
`site_id`	INTEGER FK	Site ID
`supervisor_id`	INTEGER FK	User ID who marked attendance
`date`	TEXT	Date string (YYYY-MM-DD)
`status`	TEXT	`full`, `half`, `absent`
`created_at`	DATETIME	Timestamp

Daily Site Status (`daily_site_attendance_status`)

Tracks if a site's attendance for a day is finalized.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`site_id`	INTEGER FK	Site ID
`date`	TEXT	Date string (YYYY-MM-DD)
`is_locked`	BOOLEAN	`1` if submitted/locked, `0` otherwise
`food_provided`	BOOLEAN	`1` if food was given to workers
`submitted_by`	INTEGER FK	User ID who submitted

Advances (`advances`)

Records of payments made in advance.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`labour_id`	INTEGER FK	Worker ID
`amount`	REAL	Payment amount
`date`	TEXT	Date of payment
`notes`	TEXT	Remarks

Overtime (`overtime`)

Records of extra work hours.

Column	Type	Description
`id`	INTEGER PK	Unique identifier
`labour_id`	INTEGER FK	Worker ID
`site_id`	INTEGER FK	Site ID
`hours`	REAL	Hours worked
`amount`	REAL	Calculated pay for overtime
`date`	TEXT	Date of overtime

---

2. API Reference

All endpoints are prefixed with `/api`. Most require `Authorization: Bearer <token>`.

Authentication (`/auth`)

Method	Endpoint	Description	Public?
`POST`	`/signup`	Create new Admin account	Yes
`POST`	`/signin`	Login with Phone/Password	Yes
`POST`	`/refresh-token`	Get new Access Token using Refresh Token	Yes
`POST`	`/add-supervisor`	Create a Supervisor account	No (Admin only)

Attendance (`/attendance`)

Method	Endpoint	Description
`GET`	`/`	Get attendance for a site/date (`?site_id=1&date=2023-10-27`)

POST	/	Submit daily attendance (batch). Locks the day.
GET	/summary	Get locked/submitted dates for a month (?month=10&year=2023)
GET	/lock-status	Check if a site/date is locked (is_locked: boolean)

Labours (/labours)

Method	Endpoint	Description
GET	/	List all labourers (optionally filter by site)
POST	/	Create a new labour profile
PUT	/:id	Update labour details
DELETE	/:id	Soft delete or deactivate labourer

Dashboard (/dashboard)

Method	Endpoint	Description
GET	/stats	Get admin dashboard stats (Total Labours, Active Sites, Today's Attendance count)

Reports (/reports)

Method	Endpoint	Description
GET	/salary	Generate salary report for a period. Logic: (Days * Rate) + Overtime - Advances.
GET	/attendance-summary	Aggregated attendance report per site.

Feature Logic & Business Rules

This document details the critical business logic, algorithms, and validation rules implemented in the system.

1. Attendance & Locking Mechanism

The system enforces strict rules to ensure the integrity of attendance data.

Submission Logic

- **Role**: Site Supervisors mark attendance.
- **Scope**: Attendance is marked for a specific **Site** and **Date**.
- **Batch Processing**: Supervisors submit attendance for all labourers at once.
- **Database Transaction**: The submission is wrapped in a transaction. Either all records are saved, or none are, preventing partial data states.

The "Daily Lock"

- **Trigger**: Upon successful submission of attendance for a Date/Site.
- **Effect**: A record is inserted into the daily_site_attendance_status table with is_locked = 1.
- **Consequence**: Any subsequent attempts to POST /api/attendance for that Date/Site are rejected by the backend (403 Forbidden).
- **Unlock**: Currently, there is no UI for unlocking. This requires Admin database intervention, ensuring Supervisors cannot tamper with historical data.

---

2. Wage & Salary Calculation Algorithm

The salary calculation logic is centralized in the **Reports Module** (server/routes/reports.js). It aggregates data from multiple sources to compute the "Net Payable" amount.

Core Formula

```
Net Payable = (Total Wage) + (Overtime Amount) + (Food Allowance) - (Advances)
```

Detailed Component Logic

A. Wage Calculation

The system treats the rate stored in the labours table as an **Hourly Rate**.

- **Full Day**: Calculated as **8 hours** of work.
- **Half Day**: Calculated as **4 hours** of work.
- **Absent**: 0 hours.

```
Wage = (Full_Days * 8 * Hourly_Rate) + (Half_Days * 4 * Hourly_Rate)
```

B. Food Allowance

Labourers are entitled to a monetary allowance for food if the site did **not** provide meals on a working day.

- **Check**: For every day the labourer was present (Full or Half):
    - Check daily_site_attendance_status for that Site/Date.
    - If food_provided is FALSE (or record missing), **Add Allowance**.
    - If food_provided is TRUE, **No Allowance**.
- **Amount**: Fixed at **70** currency units per applicable day.

C. Overtime

- **Source**: Sum of the amount field in the overtime table.
- **Note**: Overtime is recorded separately from standard daily attendance.

D. Advances

- **Source**: Sum of the `amount` field in the `advances` table.
- **Scope**: Advances are linked to the Labourer, not a specific site. Therefore, **all** advances for the labourer within the report period are deducted, regardless of which site they were working on.

---

3. Authentication & Security

Token Strategy

- **Access Token**: Short-lived (15 minutes). Used for API authorization (`Authorization: Bearer <token>`).
- **Refresh Token**: Long-lived (30 days). Stored securely in the `refresh_tokens` database table. Used to obtain new Access Tokens without re-login.
- **Revocation**: Logging out revokes the Refresh Token in the database.

Role-Based Access Control (RBAC)

- **Admin**:
    - Can management all data (Sites, Labours, Supervisors).
    - Can view all reports.
    - Can add new Supervisors.
- **Supervisor**:
    - Restricted to "Operational" tasks.
    - Can viewing their assigned sites (implementation pending strict filtering).
    - Can separate "manage" vs "view" permissions in future updates.

---

4. Error Handling & Edge Cases

Validation Rules

- **Future Dates**: The system rejects attendance submission for future dates (`400 Bad Request`).
- **Duplicate Phones**: User/Labour creation fails if the phone number already exists (`UNIQUE` constraint).

Concurrency Handling

- **Race Conditions**: SQLite `BEGIN IMMEDIATE` transactions (default in the driver wrapper) prevent two supervisors from locking the same site simultaneously. The first one wins; the second fails gracefully or is blocked until the first finishes.