Data Flow Documentation

This document provides a deep dive into the data flow within the application, designed to help future developers understand how data moves through the system, from the user interface to the database and back.

> ***Note**: To view the diagrams in this document, use a Markdown viewer that supports Mermaid (like GitHub, GitLab, or VS Code with the Mermaid extension). To create a PDF, you can export this Markdown file using a "Markdown to PDF" converter.*

---

1. System Overview

The application follows a standard **Client-Server-Database** data flow pattern.

- **Client (React Native)**: Initiates requests and renders data.
- **Server (Node.js/Express)**: Validates requests, executes business logic, and interacts with the database.
- **Database (SQLite)**: Stores persistent data relatonally.

Global Data Lifecycle

1. **User Interaction**: User performs an action (e.g., clicks "Submit").
2. **Client State**: React state updates; input validation occurs.
3. **API Request**: `Axios` (or custom fetch wrapper) sends an HTTP request with JSON payload.
    - *Header*: `Authorization: Bearer <token>` is attached.
4. **Server Middleware**:
    - `cors`: Allows cross-origin requests.
    - `json`: Parses body content.
    - `authenticateToken`: Verifies JWT validity.
5. **Controller Logic**:
    - Validates business rules (e.g., "Is date locked?").
    - Constructs SQL queries.
6. **Database Execution**: `sqlite3` driver executes the query (often within a transaction).
7. **Response**: Server sends JSON response (Data or Success Message).
8. **Client Update**: UI updates based on response (e.g., Toast message, list refresh).

---

2. Authentication & Session Flow

Security is handled via **JSON Web Tokens (JWT)**. There are two token types:

- **Access Token**: Short-lived (e.g., 1 hour), used for API requests.
- **Refresh Token**: Long-lived (e.g., 7 days), stored securely to obtain new access tokens.

2.1 Login Flow

```
sequenceDiagram
    participant User
    participant Client
    participant Server
    participant DB

    User->>Client: Enter Credentials (Phone/Pass)
```

```
    Client->>Server: POST /api/auth/signin
    Server->>DB: Query User (by Phone)
    DB-->>Server: User Record (Hash)
    Server->>Server: Compare Password (Bcrypt)
    alt Valid Credentials
        Server->>Server: Generate Access & Refresh Tokens
        Server-->>Client: 200 OK { user, token, refreshToken }
        Client->>Client: Store Tokens (AsyncStorage)
        Client->>User: Redirect to Home
    else Invalid
        Server-->>Client: 401 Unauthorized
        Client->>User: Show Error Toast
    end
```

2.2 Authenticated Request & Token Refresh

This flow handles automatic token refreshing when the Access Token expires.

```
sequenceDiagram
    participant Client
    participant API_Wrapper
    participant Server

    Client->>API_Wrapper: API Call (e.g., GET /labours)
    API_Wrapper->>API_Wrapper: Get Token from Storage
    API_Wrapper->>Server: Request + Auth Header

    alt Token Valid
        Server-->>API_Wrapper: 200 OK (Data)
        API_Wrapper-->>Client: Data
    else Token Expired (401/403)
        Server-->>API_Wrapper: 403 Forbidden
        API_Wrapper->>API_Wrapper: Get Refresh Token
        API_Wrapper->>Server: POST /auth/refresh-token

        alt Refresh Valid
            Server-->>API_Wrapper: 200 OK { newTokens }
            API_Wrapper->>API_Wrapper: Update Storage
            API_Wrapper->>Server: Retry Original Request
            Server-->>API_Wrapper: 200 OK (Data)
            API_Wrapper-->>Client: Data
        else Refresh Invalid
            Server-->>API_Wrapper: 403 Forbiddden
            API_Wrapper->>Client: Logout (Clear Storage)
            Client->>Client: Redirect to Login
        end
    end
```

---

3. Feature-Specific Data Flows

3.1 Attendance Marking (Bulk Action)

This is a critical flow involving bulk updates and site locking.

```
sequenceDiagram
    participant Supervisor
    participant Client
    participant Server
    participant DB

    Supervisor->>Client: Select Site & Date
    Client->>Server: GET /api/attendance/lock-status?site=X&date=Y
    Server-->>Client: { is_locked: false }

    Supervisor->>Client: Mark Attendance (P/A/H) for List
    Supervisor->>Client: Click "Submit"

    Client->>Server: POST /api/attendance
    Note right of Client: Body: { site_id, date, records: [...] }

    Server->>Server: Validate (Check if already locked)

    rect rgb(240, 248, 255)
    Note over Server, DB: Database Transaction (Begin)
    Server->>DB: Insert Attendance Records (Loop/Batch)
    Server->>DB: Insert/Update Site Status (locked=1)
    DB-->>Server: Success
    Note over Server, DB: Database Transaction (Commit)
    end

    Server-->>Client: 200 OK { message: "Attendance Locked" }
    Client->>Supervisor: Success Toast + Disable Inputs
```

3.2 Advance Payments

Recording a cash advance given to a labourer.

```
graph LR
    User[User Input] -->|Amount, Date, Notes| Client
    Client -->|POST /api/labours/:id/advance| Server

    subgraph Server Processing
        Server -->|Validate Amount| Logic[Business Logic]
        Logic -->|Insert| DB[(Advances Table)]
    end

    DB -->|Success| Server
    Server -->|201 Created| Client
    Client -->|Update State| UI[Show Updated List]
```

3.3 Overtime Submission

Recording extra hours. The server calculates the amount based on the labourer's wage rate **at the time of submission** (or current rate), ensuring historical accuracy if rates change later.

```
sequenceDiagram
    participant User
    participant Client
    participant Server
    participant DB

    User->>Client: Input Hours (e.g., 2.5)
    Client->>Server: POST /api/overtime/bulk
    Note right of Client: Body: { site_id, date, items: [{labour_id, hours}] }

    loop For Each Item
        Server->>DB: Get Labourer Rate (SELECT rate FROM labours)
        DB-->>Server: Rate (e.g., 500)
        Server->>Server: Calculate Amount ( (Rate/8) * Hours )
        Server->>DB: Insert into 'overtime' table
    end

    Server-->>Client: 200 OK
```

3.4 Salary Report Generation

How the system calculates final payable amounts.

1. **Input**: Start Date, End Date.
2. **Aggregation Queries**:
   - **Wages**: `COUNT(attendance) * Rate` (Filtered by 'Present'/'Half').
   - **Overtime**: `SUM(overtime.amount)` .
   - **Advances**: `SUM(advances.amount)` .
3. **Calculation**: `Total = Wages + Overtime - Advances` .

```
classDiagram
    class ReportService {
        +getSalaryReport(startDate, endDate)
    }
    class Calculations {
        +calculateWages(days, rate)
        +sumAdvances(list)
        +sumOvertime(list)
    }

    ReportService --> Calculations : Uses
    ReportService ..> Database : Aggregation SQL
```

---

4. Error Handling & Edge Cases

4.1 Network Errors

- **Scenario**: User is offline or server is down.
- **Flow**:
    1. `fetch` throws exception.
    2. `api.ts` catches error.
    3. Client displays generic "Network Error" or "Please check connection".
    4. **No Offline Mode**: Currently, the app requires internet to function. Data is not queued locally (V1 limitation).

4.2 Data Integrity Locks

- **Scenario**: Two supervisors mark attendance for the same site same day.
- **Protection**:
    - Database constraint on `daily_site_attendance_status (site_id, date)`.
    - Transaction isolation prevents partial writes.
    - Once `locked=1`, the API rejects further POST requests for that date/site.

4.3 Validation

- **Frontend**: React logic prevents submitting empty forms or negative numbers.
- **Backend**: `express-validator` or manual checks ensure inputs are sanitized and valid types before hitting the DB.

---

5. Database Relationships (ERD Reference)

A simplified view of how data connects to support the flows above.

```
erDiagram
    USERS ||--o{ SITES : manages
    SITES ||--o{ LABOURS : assigned_to
    LABOURS ||--o{ ATTENDANCE : has
    LABOURS ||--o{ ADVANCES : receives
    LABOURS ||--o{ OVERTIME : performs
    SITES ||--o{ DAILY_STATUS : tracks

    USERS {
        string role "Admin/Supervisor"
    }
    DAILY_STATUS {
        boolean is_locked "Prevents edits"
    }
```