**Project Introduction Document**

**Project Title:**

Vendor Payment Tracking System

**Overview**

This project aims to develop a Vendor Payment Tracking System, a mobile and desktop application for tracking partial payments between small vendors and their clients (middlemen) and the associated factory owners. Designed for business scenarios where installments are often paid manually, this app will allow users to log payments without integrating a payment gateway, ensuring small vendors and their clients can easily manage and reconcile transactions.

The system targets middlemen and factory owners in industries like small-scale manufacturing (e.g., car seat covers), where payment tracking is often manual and prone to errors. By offering a structured, accessible way to log and verify payments, both parties can stay informed on payment statuses, due dates, and balances, enhancing trust and reducing potential discrepancies.

**Goal**

To provide a user-friendly platform for tracking partial payments, logging installment records, and validating payment details between small vendors and their clients.

**Scope**

The application will focus solely on tracking payments, not processing them. It will allow users to manually log and verify payments made through external platforms.

**User Roles**

1. **Middleman (Client)**
   * Responsible for managing orders placed with the factory owner and logging payment installments.
2. **Factory Owner**
   * Oversees orders and payment logs submitted by middlemen, confirming and validating each payment for accuracy.

**Core Features**

**a. User Registration and Login**

* **Secure Login and Authentication**: Supports secure, role-based login and authentication for middlemen and factory owners.
* **Role-Based Registration**: Users select their role (Middleman or Factory Owner) during registration, granting access to tailored functionalities.
* **Profile Management**: Allows users to add business details, contact information, and other relevant data.

**b. Order Management**

* **Order Creation**: Middlemen can create orders, detailing specifics such as product details, total amount, due dates, and notes.
* **Order Approval**: When a middleman creates an order, it will appear as a pending order in the factory owner's dashboard. The factory owner can then accept or reject the order:
  + If Accepted: The order is visible to both middleman and factory owner.
  + If Rejected: The order is dismissed, and the middleman is notified.

c**. Payment Tracking (Hisab Management)**

* **Installment Logging**: Middlemen can record partial payments made externally and log them as installments.
* **Verification**: Factory owners can view and confirm payments for each order to avoid discrepancies.
* **Payment Details**: Logs include:
  + Date and time of payment
  + Amount paid
  + Remaining balance

**d. Notifications and Alerts**

* **Reminders for Due Payments**: Middlemen receive reminders when a payment is due.
* **Order Status Updates**: Factory owners receive notifications when new orders are created or payments are logged.
* **Confirmation Alerts**: Both users receive updates when a payment is verified.

**e. Dashboard Overview**

* **Dashboard Components:**
  + A summary of ongoing orders, their payment statuses, and installment history.
  + Filtering and sorting options to help users quickly find specific orders or transactions.

f**. Reports and Statements**

* Transaction Reports: Users can generate monthly or custom-period reports.
* Export Options: Reports are downloadable as PDFs or Excel files for offline reference.

**Application Architecture**

Frontend: Mobile & Desktop

* Technology:
  + Mobile: React Native for iOS and Android compatibility.
  + Desktop: Electron for creating a cross-platform desktop app.
* UI Library: Material UI or similar, ensuring a consistent user interface.

Backend

* Framework: Node.js with Express for request handling.
* Database: MongoDB for storing profiles, orders, payments, and verification status.
* APIs: RESTful API for seamless data exchange between frontend and backend.
* Authentication: JWT-based secure authentication, with middleware for role-based access control.

High-Level User Flow

1. Registration and Login
   * Users select a role (Middleman or Factory Owner) and are directed to a tailored dashboard after login.
2. Order Creation & Approval
   * Middlemen create orders, which factory owners review and approve or reject.
3. Payment Logging and Confirmation
   * Middlemen log payments; factory owners confirm them, keeping both parties informed.
4. Tracking and Reminders
   * Dashboards display ongoing orders and payment reminders.
5. Report Generation
   * Users can generate detailed reports for tracking and reconciliation purposes.

Development Phases

1. Phase 1: Planning and Design
   * Requirements gathering, database schema design, UI wireframing.
2. Phase 2: Frontend and Backend Setup
   * Initialize React Native and Electron environments and set up core backend functionality.
3. Phase 3: Feature Development
   * Implement user registration, order and payment tracking, and dashboards.
4. Phase 4: Testing and QA
   * Conduct user testing, security checks, and validation of payment tracking functionality.
5. Phase 5: Deployment and Feedback
   * Deploy the app, collect user feedback, and iterate for improvements.

Conclusion

The Vendor Payment Tracking System will offer a streamlined approach to tracking and managing partial payments for small vendors and their clients, ensuring transparency, accuracy, and trust. This system is an accessible, cost-effective solution for businesses that rely on installment-based payments, bringing structure to manual, external payments without requiring a payment gateway integration.

**Database Design Document**

**Project**: Payment Tracking System for Small Vendors  
**Database Type**: SQL

**Overview**

This database is designed to manage payment tracking between middlemen (vendors) and factory owners. It stores information about users, orders, payment instalments, and order status updates, facilitating clear tracking of payment and order statuses. Authentication is based on phone numbers, with OTPs sent for verification, making email optional.

**Table of Contents**

1. [Users Table](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#1-users-table)
2. [Orders Table](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#2-orders-table)
3. [Payments Table](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#3-payments-table)
4. [Order Status Updates Table](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#4-order-status-updates-table)
5. [OTP Verifications Table](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#5-otp-verifications-table-optional)
6. [Notifications Table (Optional)](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#6-notifications-table-optional)
7. [ER Diagram Overview](https://chatgpt.com/c/6734e62d-29e8-800f-9db8-62207899d1cc#er-diagram-overview)

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| user\_id | SERIAL | PRIMARY KEY | Unique identifier for each user. |
| username | VARCHAR(50) | NOT NULL | Username for login and display purposes. |
| email | VARCHAR(100) | UNIQUE, NULLABLE | Email address (optional for small vendors). |
| phone\_number | VARCHAR(15) | UNIQUE, NOT NULL | Primary contact number for OTPs and login. |
| password | VARCHAR(255) | NOT NULL | Hashed password for authentication. |
| role | ENUM('middleman', 'factory\_owner') | NOT NULL | Defines user type: "middleman" or "factory\_owner". |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp for account creation. |

**1. Users Table**

Stores information about all users, both middlemen and factory owners.

**2. Orders Table**

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| order\_id | SERIAL | PRIMARY KEY | Unique identifier for each order. |
| middleman\_id | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the middleman who created the order. |
| factory\_owner\_id | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the associated factory owner. |
| total\_amount | DECIMAL(10, 2) | NOT NULL | Total payment amount for the order. |
| description | TEXT | NULLABLE | Description of the order (e.g., item details). |
| status | ENUM('pending', 'accepted', 'rejected') | DEFAULT 'pending' | Current status of the order. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp for when the order was created. |

Stores information about orders created by middlemen for factory owners.

**3. Payments Table**

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| payment\_id | SERIAL | PRIMARY KEY | Unique identifier for each payment entry. |
| order\_id | INT | FOREIGN KEY REFERENCES Orders(order\_id) | ID of the associated order. |
| paid\_by | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the middleman making the payment. |
| amount | DECIMAL(10, 2) | NOT NULL | Amount paid in this installment. |
| payment\_date | DATE | NOT NULL | Date when the payment was made. |
| status | ENUM('pending', 'confirmed') | DEFAULT 'pending' | Status of the payment (e.g., middleman-entered, confirmed by owner). |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp for when the payment entry was created. |

Records installment payments made by middlemen toward specific orders.

**4. Order Status Updates Table**

Tracks changes in the order status to maintain a history of status updates.

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| status\_update\_id | SERIAL | PRIMARY KEY | Unique identifier for each status update. |
| order\_id | INT | FOREIGN KEY REFERENCES Orders(order\_id) | ID of the order being updated. |
| status | ENUM('accepted', 'rejected') | NOT NULL | Updated status of the order. |
| updated\_by | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the user who made the update. |
| updated\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp of the status update. |

**5. OTP Verifications Table (Optional)**

Stores OTPs for phone-based authentication and password resets.

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| otp\_id | SERIAL | PRIMARY KEY | Unique identifier for each OTP entry. |
| user\_id | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the user requesting OTP. |
| otp\_code | VARCHAR(6) | NOT NULL | The OTP sent to the user. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp for OTP creation. |
| expires\_at | TIMESTAMP | NOT NULL | Expiration time for the OTP. |

**6. Notifications Table (Optional)**

Stores notifications for users about actions like new orders, payments, or status updates.

| **Column** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| notification\_id | SERIAL | PRIMARY KEY | Unique identifier for each notification. |
| user\_id | INT | FOREIGN KEY REFERENCES Users(user\_id) | ID of the user receiving the notification. |
| message | TEXT | NOT NULL | Notification message text. |
| is\_read | BOOLEAN | DEFAULT FALSE | Indicates if the notification has been read. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp for notification creation. |

**ER Diagram Overview**

1. **Users**:
   * **Orders** - One-to-many relationship (middleman to orders).
   * **Payments** - One-to-many relationship (middleman to payments).
   * **Order Status Updates** - One-to-many relationship (factory owner updates on orders).
   * **OTP Verifications** - One-to-one relationship for OTP verification (optional).
2. **Orders**:
   * **Payments** - One-to-many relationship (each order can have multiple payments).
   * **Order Status Updates** - One-to-many relationship (each order can have multiple status updates).
3. **Notifications**:
   * Independent table used to alert users to specific updates.

**Summary**

This schema provides a comprehensive structure for managing user roles, order tracking, payment installments, and OTP-based authentication without email. The setup is flexible to allow for feature expansion, such as adding notifications or tracking payment histories.

**1. Backend Stack Selection**

* **Language**: JavaScript/TypeScript
* **Framework**: Express (Node.js)
* **Database**: SQL (e.g., PostgreSQL, MySQL)
* **ORM**: Sequelize or Knex.js (if you want an abstraction layer)
* **Authentication**: JWT (for session management) with OTP verification for phone numbers.

**2. File Structure**

A modular file structure makes the codebase organized and scalable. Here’s a suggested structure:

**3. Routing and API Endpoints**

Designing routes that align with the RESTful principles for clear, predictable APIs.

**a. Authentication Routes (**authRoutes.js**)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /register | Register a new user |
| POST | /login | Log in a user with OTP |
| POST | /otp/verify | Verify OTP for login |
| POST | /password/reset | Reset password with OTP |

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /user/:id | Get user details |
| PUT | /user/:id/update | Update user profile |

**b. User Routes (**userRoutes.js**)**

**c. Order Routes (**orderRoutes.js**)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /order | Create a new order (middleman only) |
| GET | /orders | Get all orders (middleman or factory owner) |
| GET | /order/:id | Get a specific order by ID |
| PUT | /order/:id/status | Update order status (e.g., accepted, rejected) |

**d. Payment Routes (**paymentRoutes.js**)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /payment | Add a new payment installment to an order |
| GET | /payments/order/:id | Get all payments for a specific order |
| PUT | /payment/:id/status | Confirm payment (factory owner only) |
|  |  |  |

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| GET | /api/users/profile | Fetch profile details for the logged-in vendor |

|  |  |  |
| --- | --- | --- |
| GET | /api/suppliers | Get a list of suppliers with active orders |

|  |  |  |
| --- | --- | --- |
| GET | /api/orders | Fetch all orders for the vendor |

|  |  |  |
| --- | --- | --- |
| GET | /api/orders/:order\_id | Fetch details of a specific order |

|  |  |  |
| --- | --- | --- |
| POST | /api/orders | Create a new order |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| PUT | /api/orders/:order\_id/status | Update the status of an order |

|  |  |  |
| --- | --- | --- |
| POST | /api/payments | Add a payment for a specific order |

  "username": "Vishesh",

  "phone\_number": "9555830578",

  "password": "V12345678",

  "role": "Vendor"

 "username": "sahil",

  "email":"sahilshukla20034@gmail.com",

  "phone\_number": "9555830579",

  "password": "S12345678",

  "role": "Supplier"