Electricity Billing System
Documentation & Test Procedures

1. Backend Architecture (`/backend`)

The backend is built on **Node.js** and **Express**, utilizing **MongoDB** (via Mongoose) for data persistence. It adheres to a RESTful architecture with stateless authentication via **Json Web Token (JWT)**.

1.1 Server Entry Point (`/server.js`)

This file initializes the application, connects to the database, and routes incoming HTTP requests.

`initializeAdmin()` (Internal Async Function)

- **Trigger:** Called immediately after the MongoDB connection is established.
- **Purpose:** Ensures a default Admin account exists to bootstrap the system.
- **Logic:**
    1. Queries the `User` collection for `{ role: "admin" }`.
    2. If not found, creates a user with:
        - **Email:** `admin@test.com`
        - **Password:** `admin1` (hashed by User model hook)
        - **Role:** `admin`

Middleware

These functions intercept requests before they reach the route handlers.

1. `protect(req, res, next)`

- **Purpose:** Authentication. Validates the JWT.
- **Required Header:** `Authorization: Bearer <token>`
- **Logic:**
    1. Checks if the header exists and starts with "Bearer".
    2. Extracts the token string.
    3. Calls `jwt.verify(token, JWT_SECRET)` to decode the payload.
    4. Uses `decoded.id` to fetch the user from the DB (excluding password).
    5. Attaches the user document to `req.user`.

- **Error Handling:** Returns `401 Not Authorized` if the token is missing, invalid, or expired.

2. `admin(req, res, next)`

- **Purpose:** Authorization (Role-based Access Control).
- **Logic:** Checks if `req.user.role === 'admin'`.
- **Error:** Returns `403 Forbidden` if there is a role mismatch.

3. `employee(req, res, next)`

- **Purpose:** Authorization.
- **Logic:** Allows access if `req.user.role` is either `'employee'` or `'admin'`.

API Endpoints (Routes)

1. POST `/api/auth/login`

- **Access:** Public
- **Request Body:**

  JSON

  ```
  { "email": "user@test.com", "password": "password123", "role": "user" }
  ```

- **Logic:**
  1. Finds user by `email`.
  2. Calls `user.matchPassword(password)`.
  3. If `role` is provided, verifies `user.role` matches.
- **Response (Success 200):** Returns User ID, Name, JWT Token, and Role.

2. POST `/api/bills/create`

- **Access:** Protected (`protect`, `employee`)
- **Request Body:**

  JSON

  ```
  { "serviceNo": "100001", "currentReading": 1500, "employeeZone": "North" }
  ```

- **Logic Flow:**
  1. **Lookup:** Finds Consumer by `serviceNo`. Returns 404 if not found.
  2. **Validation:** Checks if `currentReading` >= `consumer.meterReading`. Returns 400 if invalid.
  3. **Calculation:** Calls `calculateBill` (see Utilities).

4. **Persistence:**
    - Creates a new `Bill` document with status 'Pending'.
    - Updates the `Consumer` document's `meterReading`.

- **Response (Success 201):** Returns the created Bill object.

1.2 Data Models (`/models`)

User.js Represents all system actors (Consumers, Employees, Admins).

- **Schema Fields:**
    - `name`: String (Title Case enforced).
    - `email`: String (Unique, Validated via Regex).
    - `password`: String (Bcrypt Hash).
    - `role`: Enum [`admin`, `employee`, `user`].
    - `serviceNo`: String (Unique, Auto-generated).
    - `category`: Enum [`household`, `commercial`, `industry`] (Required for users).
- **`pre('save')` Hook:**
    - **Trigger:** Runs before `user.save()` or `User.create()`.
    - **Task 1 (Name Formatting):** Converts "vinith kumar" -> "Vinith Kumar".
    - **Task 2 (Service ID Generation):**
        - Generates a prefix based on category (Household="1", Commercial="2", Industry="3").
        - Increments a global Counter.
        - Format: `{Prefix}{Count padded to 6 digits}` (e.g., "1000042").
    - **Task 3 (Security):** Hashes the password using a 10-round salt.

Bill.js Represents a generated electricity bill.

- **Schema Fields:**
    - `user`: ObjectId Reference -> User.
    - `units`: Number (Calculated consumption).
    - `billAmount`: Number (Energy Charge).
    - `fineAmount`: Number (Fixed 150 if arrears existed).

- `totalAmount`: Number (Bill + Fine + Arrears).
- `status`: Enum [`Paid`, `Pending`].

1.3 Utilities (`/utils`)

`billCalculator.js` Encapsulates the domain logic for billing.

Function: `calculateBill(userCategory, units, lastBill)`

- **Parameters:**
  - `units`: Calculated as (`currentReading` - `prevReading`).
  - `lastBill`: The most recent bill document (used to find unpaid dues).
- **Algorithm:**
  - **Energy Charge:**
    - 0-50 Units: × 1.5
    - 51-100 Units: × 2.5
    - 101-150 Units: × 3.5
      - 150 Units: × 4.5
  - **Arrears Check:**
    - If `lastBill` exists AND `lastBill.dueAmount > 0`:
      - Adds `previousDue`.
      - Adds `fineAmount` (150).
  - **Total:** Energy Charge + Previous Due + Fine.

---

2. Frontend Architecture (`/frontend`)

The frontend uses Vanilla JavaScript with `localStorage` for session management.

2.1 Global Logic (`main.js`)

- **`getAuthHeaders()`**: Centralized function that returns headers including `Content-Type` and the `Authorization` Bearer token.
- **`handleLogin()`**:
  1. Extracts email/password from the DOM.
  2. Sends a `fetch` POST request to the backend.

3. **On Success:** Stores the user object in `localStorage` and redirects based on role:
    - Admin -> `admin_dashboard.html`
    - Employee -> `employee_dashboard.html`
    - User -> `user_dashboard.html`

2.2 User Dashboard (`user.js`)

- **`loadBills()`**:
    1. Fetches `GET /api/bills/my-bills`.
    2. Iterates through the bills array.
    3. If status is 'Pending', renders inputs for Transaction ID and Amount.
    4. Injects HTML into `#billArea`.
- **`payBill(billId)`**:
    1. Reads Transaction ID and Amount from the specific bill card.
    2. Sends `fetch POST /pay-bill`.
    3. **On Success:** Alerts "Payment successful" and refreshes the list to update status to "Paid" (Green).

2.3 Employee Dashboard (`employee.js`)

- **`generateBill()`**:
    1. User enters Service No and Current Reading.
    2. Sends `fetch POST /api/bills/create`.
    3. **On Success:** Displays a "Receipt Preview" HTML block showing Arrears, Fine, and Total.

---

3. Comprehensive Test Case

This test case verifies the complete workflow of the Electricity Billing System, ensuring all roles (Admin, Employee, User), authorization checks, database updates, and calculations work correctly in sequence.

Prerequisites:

- Database is running (`mongod`).
- Server is running on port 5200.
- Default Admin account exists (`admin@test.com` / `admin1`).

Step 1: Admin Setup (Employee Registration)

Actor: Admin **Goal:** Create an Employee account responsible for the "North" zone.

1. **Login:**
    - Input: `admin@test.com` / `admin1`. Role: Admin.
    - *Expected:* Redirect to `admin_dashboard.html`.
2. **Action:** Click "Add Employee".
    - Input: Name: `North Lineman`, Email: north`@emp.com`, Zone: `North`, Password: `password`.
    - Click Submit.
    - *Expected:* Alert "Employee Registered". Database now contains this employee.

Step 2: Consumer Registration

Actor: New User (Consumer) **Goal:** Register a new household connection.

1. **Action:** Open `index.html` -> Click "Register".
2. **Input:**
    - Name: `vinith` (Lowercase input to test auto-formatting).
    - Email: `vinith@home.com`, Mobile: `9998887776`.
    - Category: `Household`, Zone: `North` (Must match Employee's zone).
    - Password: `user123`.
3. **Submit:**
    - *Expected:* Alert showing Service No (e.g., `1000001`).
    - *Verify:* Name is saved in DB as **Vinith** (Title Case).

Step 3: Bill Generation

Actor: Employee (John Lineman) **Goal:** Generate a bill for the new consumer.

1. **Login:** Input `john@power.com` / `password123` / Employee.
    - *Expected:* Redirect to `employee_dashboard.html`. Zone shows "North".
2. **Action:** Generate Bill.
    - Input: Service No: `1000001`, Current Reading: `100` (Assumes previous was 0).
    - Click Submit.
3. **Expected Result:**
    - Receipt Preview appears.

- **Calculation Check:**
    - Units: 100
    - Tier 1 (0-50 * 1.5): 75
    - Tier 2 (50-100 * 2.5): 125
    - **Total Bill Amount: 200**
    - Fine/Arrears: 0

Step 4: Bill Payment

Actor: Consumer (Vinith) **Goal:** View the detailed bill and pay it.

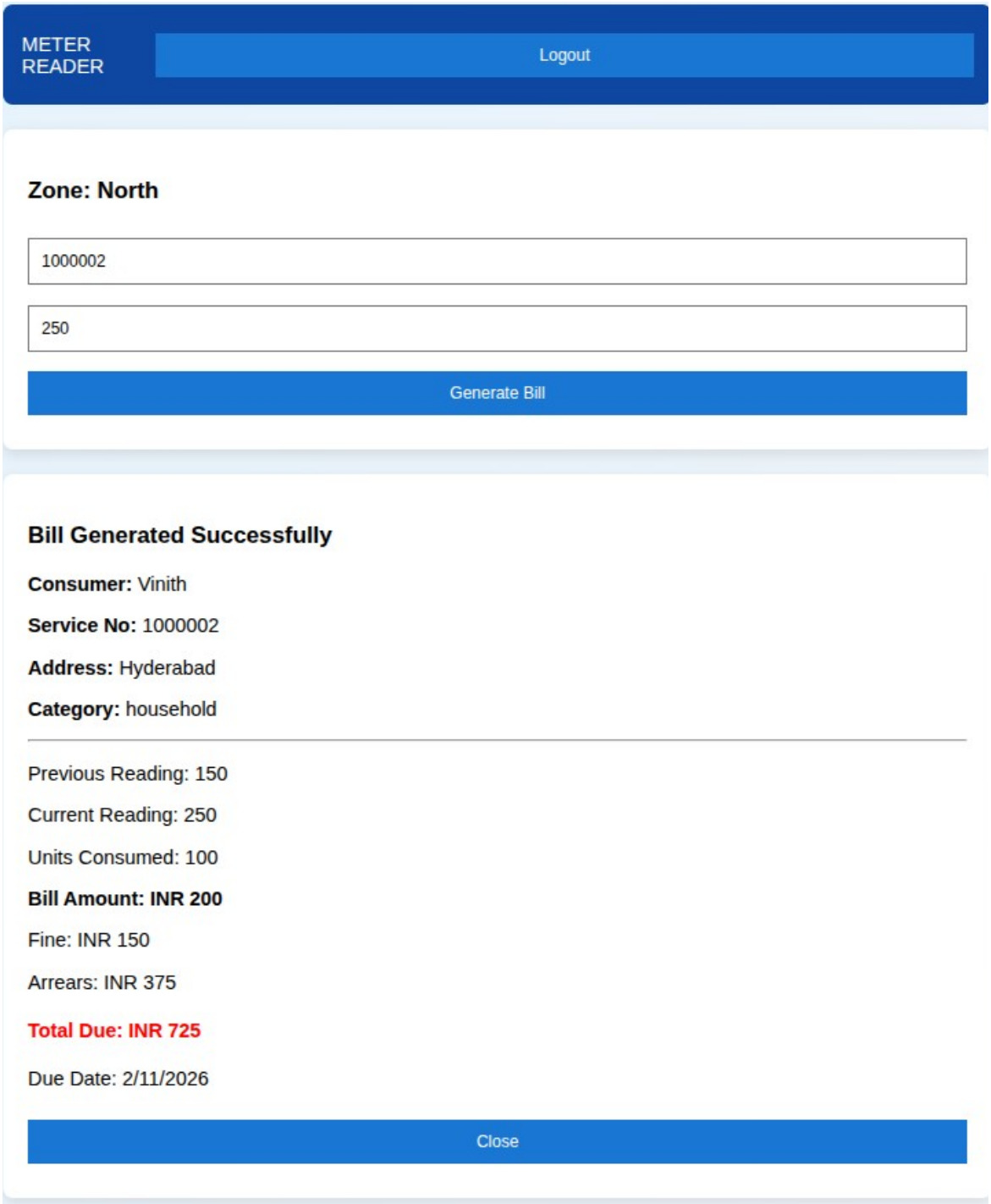
1. **Login:** Input `vinith@gmail.com` / Consumer.
    - *Expected:* Redirect to `user_dashboard.html`.
2. **Verify Dashboard:**
    - Profile shows Service No `1000001`.
    - One Bill Card is visible. Status: **Pending (Red)**. Due Amount: **200**.
3. **Action:** Pay Bill.
    - Input Transaction ID: `TXN12345`, Amount: `200`.
    - Click Pay.
    - *Expected:* Alert "Payment successful". Page refreshes. Status updates to **Paid (Green)**. Due Amount: 0.

Step 5: Verification of Updates

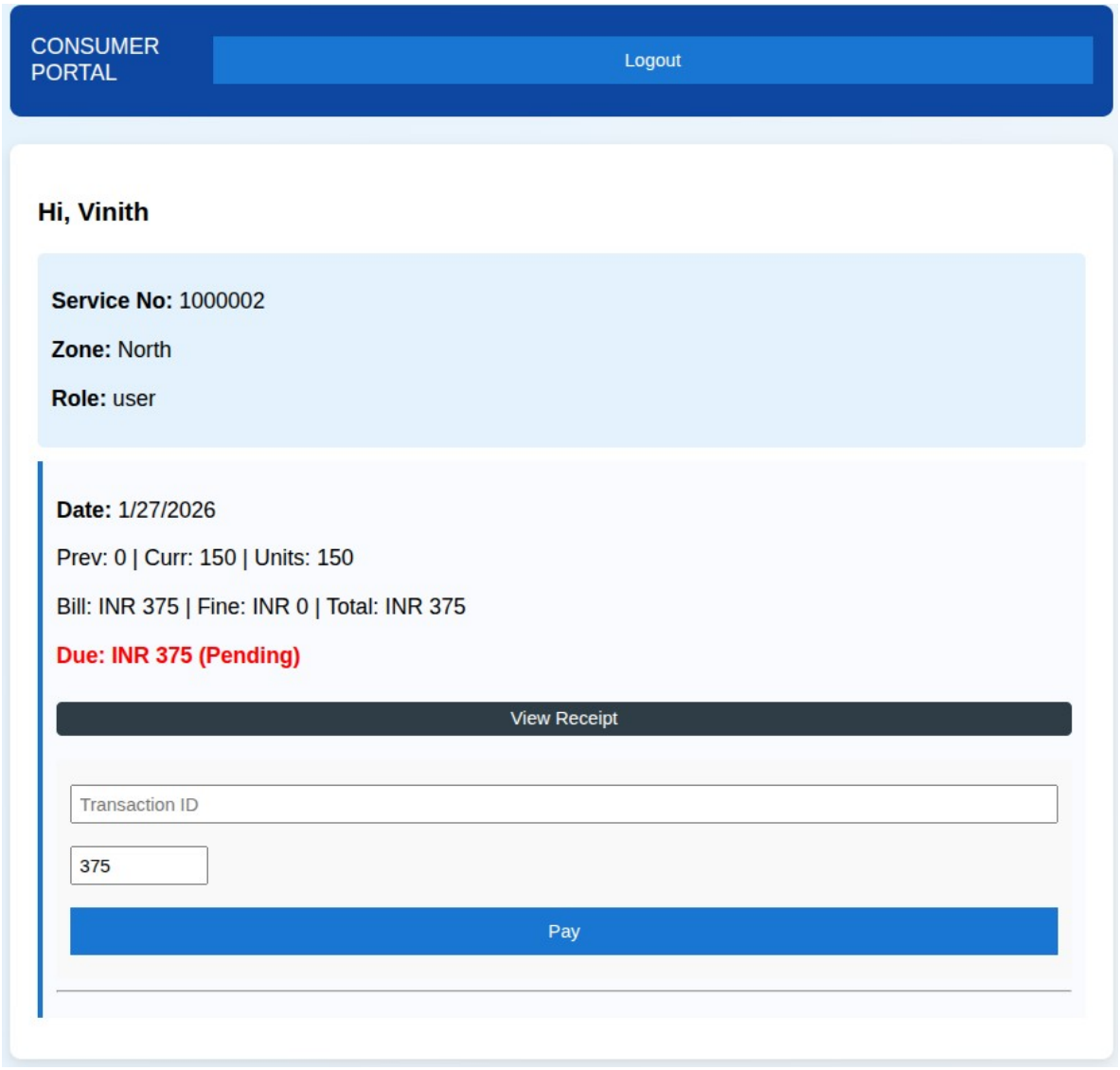
Actor: System / Admin **Goal:** Ensure system state is consistent.

1. **Check User Meter:** User `meterReading` in the database should now be `100`.
2. **Check Admin View:** Login as Admin -> Click "Consumers" -> Find Vinith. Total Due should be `0`

Generated Bill by Employee



Consumer Portal and list of bills



Click on view reciept you can see the detailed Bill like below.

Recipt of every month Bill

TS NPDCL

tsnpdcl@gmail.com
95674 64321 | www.tsnpdcl.com

Electricity Payment Receipt

Receipt No.:	1000002-2026
Date of Payment:	1/27/2026
Customer Name:	Vinith
Category / Zone:	undefined / North
Amount Paid:	INR 0
Payment Method:	Online
Invoice No.:	214E326D

Billing Period:	2/11/2026 - 1/27/2026
Energy Consumed:	150 kWh
Rate per kWh:	INR 2.50
Total Consumption Charge:	INR 375
Fine / Penalty:	INR 0
Arrears / Due Amount:	INR 375
Total Payable Amount:	INR 375
Taxes:	INR 0.00

This receipt confirms payment for electricity services. Thank you for your payment!

TSNPDCL - TELANGANA

Authorized Signature: _____

Back to Dashboard