 **login.js**: Handles login functionality and redirects to the dashboard.

 **manage-users.js**: Manages users including CRUD operations.

 **manage-patients.js**: Manages patients including CRUD operations.

 **manage-appointments.js**: Manages appointments including CRUD operations.

 **billing.js**: Manages billing and generates bills.

 **reports.js**: Generates and displays reports.

 **settings.js**: Manages system settings.

 **login.html**: Allows patients to log in.

 **register.html**: Allows patients to register.

 **dashboard.html**: Main page for patients with access to their bills.

 **view-bills.html**: Shows all bills and allows payments.

 **change-password.html**: Lets patients change their passwords.

 **logout.html**: Redir

To create a database and integrate all the functionalities, you'll need to follow several steps, including setting up the database schema, creating necessary tables, and implementing server-side code to handle the interactions between your frontend and database. I'll guide you through this process step-by-step.

**1. Database Schema**

Assuming you are using MySQL or a similar SQL database, here's how you can define your database schema:

**Create Database**

sql

Copy code

CREATE DATABASE hms;

USE hms;

**Create Tables**

**1. Patients Table**

sql

Copy code

CREATE TABLE patients (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL

);

**2. Bills Table**

sql

Copy code

CREATE TABLE bills (

id INT AUTO\_INCREMENT PRIMARY KEY,

patient\_id INT,

date DATE,

amount DECIMAL(10, 2),

status ENUM('Unpaid', 'Paid') DEFAULT 'Unpaid',

FOREIGN KEY (patient\_id) REFERENCES patients(id)

);

**3. Appointments Table** (optional, if needed for appointment functionality)

sql

Copy code

CREATE TABLE appointments (

id INT AUTO\_INCREMENT PRIMARY KEY,

patient\_id INT,

date DATE,

time TIME,

description TEXT,

FOREIGN KEY (patient\_id) REFERENCES patients(id)

);

**2. Server-Side Code**

You can use Node.js with Express and a MySQL database for your server-side code. Here’s an overview of the server-side implementation for the functionalities:

**Setup**

1. **Install Dependencies**

bash

Copy code

npm install express mysql2 body-parser bcryptjs jsonwebtoken

1. **Create server.js**

Here’s a basic implementation for handling patient registration, login, bill viewing, and payment:

javascript

Copy code

// server.js

const express = require('express');

const mysql = require('mysql2');

const bodyParser = require('body-parser');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const app = express();

const port = 3000;

// Database connection

const db = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'password',

database: 'hms'

});

db.connect(err => {

if (err) throw err;

console.log('Connected to database');

});

// Middleware

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

// Secret key for JWT

const JWT\_SECRET = 'your\_jwt\_secret\_key';

// Register patient

app.post('/api/patients/register', (req, res) => {

const { username, password, email } = req.body;

if (!username || !password || !email) return res.status(400).json({ success: false, message: 'All fields are required' });

bcrypt.hash(password, 10, (err, hashedPassword) => {

if (err) return res.status(500).json({ success: false, message: 'Error hashing password' });

const sql = 'INSERT INTO patients (username, password, email) VALUES (?, ?, ?)';

db.query(sql, [username, hashedPassword, email], (err, result) => {

if (err) return res.status(500).json({ success: false, message: 'Error registering patient' });

res.status(201).json({ success: true, message: 'Registration successful' });

});

});

});

// Login patient

app.post('/api/patients/login', (req, res) => {

const { username, password } = req.body;

if (!username || !password) return res.status(400).json({ success: false, message: 'All fields are required' });

const sql = 'SELECT \* FROM patients WHERE username = ?';

db.query(sql, [username], (err, results) => {

if (err) return res.status(500).json({ success: false, message: 'Error querying database' });

if (results.length === 0) return res.status(401).json({ success: false, message: 'Invalid credentials' });

const patient = results[0];

bcrypt.compare(password, patient.password, (err, isMatch) => {

if (err) return res.status(500).json({ success: false, message: 'Error comparing passwords' });

if (!isMatch) return res.status(401).json({ success: false, message: 'Invalid credentials' });

const token = jwt.sign({ id: patient.id }, JWT\_SECRET, { expiresIn: '1h' });

res.json({ success: true, token, patientId: patient.id });

});

});

});

// Middleware to verify token

const authenticateToken = (req, res, next) => {

const token = req.headers['authorization'];

if (!token) return res.status(401).json({ success: false, message: 'Access denied' });

jwt.verify(token, JWT\_SECRET, (err, user) => {

if (err) return res.status(403).json({ success: false, message: 'Invalid token' });

req.user = user;

next();

});

};

// Get bills

app.get('/api/patients/:id/bills', authenticateToken, (req, res) => {

const patientId = req.params.id;

const sql = 'SELECT \* FROM bills WHERE patient\_id = ?';

db.query(sql, [patientId], (err, results) => {

if (err) return res.status(500).json({ success: false, message: 'Error querying bills' });

res.json({ success: true, bills: results });

});

});

// Pay bill

app.post('/api/bills/:id/pay', authenticateToken, (req, res) => {

const billId = req.params.id;

const { paymentMethod } = req.body;

if (!paymentMethod) return res.status(400).json({ success: false, message: 'Payment method is required' });

const sql = 'UPDATE bills SET status = "Paid" WHERE id = ?';

db.query(sql, [billId], (err, result) => {

if (err) return res.status(500).json({ success: false, message: 'Error updating bill' });

res.json({ success: true, message: 'Payment successful' });

});

});

// Change password

app.post('/api/patients/:id/change-password', authenticateToken, (req, res) => {

const patientId = req.params.id;

const { currentPassword, newPassword } = req.body;

if (!currentPassword || !newPassword) return res.status(400).json({ success: false, message: 'All fields are required' });

const sql = 'SELECT password FROM patients WHERE id = ?';

db.query(sql, [patientId], (err, results) => {

if (err) return res.status(500).json({ success: false, message: 'Error querying patient' });

if (results.length === 0) return res.status(404).json({ success: false, message: 'Patient not found' });

const patient = results[0];

bcrypt.compare(currentPassword, patient.password, (err, isMatch) => {

if (err) return res.status(500).json({ success: false, message: 'Error comparing passwords' });

if (!isMatch) return res.status(401).json({ success: false, message: 'Current password is incorrect' });

bcrypt.hash(newPassword, 10, (err, hashedPassword) => {

if (err) return res.status(500).json({ success: false, message: 'Error hashing new password' });

const sql = 'UPDATE patients SET password = ? WHERE id = ?';

db.query(sql, [hashedPassword, patientId], (err, result) => {

if (err) return res.status(500).json({ success: false, message: 'Error updating password' });

res.json({ success: true, message: 'Password changed successfully' });

});

});

});

});

});

// Start server

app.listen(port, () => {

console.log(`Server running on http://localhost:${port}`);

});

**Summary**

1. **Database Schema**: Created tables for patients, bills, and optionally appointments.
2. **Server-Side Code**:
   * **Registration**: Hashes passwords and stores user information.
   * **Login**: Authenticates users and issues JWT tokens.
   * **Get Bills**: Retrieves bills for a specific patient.
   * **Pay Bill**: Updates the status of a bill to "Paid".
   * **Change Password**: Allows patients to change their passwords securely.

Adjust the endpoints and database details according to your actu