**A PROPOSED OFFERING OF A CLINIC RECORDS MANAGEMENT SYSTEM**

**FOR**

**HI-PRECISION DIAGNOSTICS – MALABON BRANCH**

A Thesis Project Presented to the

Faculty of Datamex College of Saint Adeline, Inc.

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in Information Technology

By:

Catubay, Mark Lawrence L.

**TECHNICAL DOCUMENT**

**CHAPTER I**

**INTRODUCTION**

**Purpose of the Document**

This document provides a comprehensive technical overview of the Clinic Records Management System (CRMS). It is intended for software developers, system administrators, and technical staff responsible for the installation, configuration, maintenance, and future development of the application.

**System Overview**

The CRMS is a standalone desktop application designed to digitize and manage patient records, consultation histories, and medicine inventory for the Hi-Precision Diagnostics – Malabon Branch.

**Scope of the Documentation**

This document covers the system architecture, installation procedures for a development environment, API specifications, database schema, code structure, and maintenance guidelines.

**CHAPTER II**

**SYSTEM OVERVIEW**

**System Architecture**

The CRMS is built on a three-tier architecture packaged within a single desktop application container.

* + Presentation Layer (Frontend): A Multi-Page Application (MPA) built with standard HTML5, CSS3, and vanilla JavaScript. It is responsible for all user interface rendering and client-side interactions.
  + Logic Layer (Backend): A RESTful API server built with Node.js and the Express.js framework. It handles all business logic, data validation, and database communication.
  + Data Layer (Database): A self-contained SQLite database engine that stores all application data in a single file (clinic.db).

**High-Level Components**

The primary components are the Electron main process, which creates the native application window, and the packaged Node.js server, which runs in the background. The frontend (loaded in the Electron window) communicates with the backend via local HTTP requests.

**Deployment Architecture**

The entire system is packaged using electron-builder into a single Windows installer executable (.exe). This creates a fully offline, standalone application that does not require any external dependencies (like Node.js or a web server) to be installed on the end-user's machine.

**CHAPTER III**

**INSTALLATION GUIDE**

This guide is for developers setting up the project on their local machine for development or maintenance purposes.

* System Requirements:
  + OS: Windows 10/11, macOS, or Linux
  + Software:
    - Node.js (v18.x or later)
    - npm (v9.x or later)
    - Git
    - A code editor such as Visual Studio Code
* Step-by-Step Installation:
  + Clone the Repository: Open a terminal or command prompt and clone the project source code: **git clone https://github.com/MarkLawrenceArtistry/crms.git**
  + Navigate to Project Directory: **cd crms-thesis**
  + Install Dependencies: Install all required Node.js packages listed in package.json: **npm install**
  + Run the Application: Start the application in development mode. This will launch the Electron window**. npm start**
  + Build the Installer: To create the distributable .exe installer for deployment, run the build script: **npm run dist**
  + The output will be located in the /dist directory. Search for the Installer exe.

**CHAPTER IV**

**CONFIGURATION GUIDE**

The CRMS is designed for near-zero configuration on the end-user's machine. However, developers can modify the following settings in the source code.

* **Server Port:** The default port is defined in app.js. While this is less critical inside the Electron environment, it is used for local development.
  + File: app.js
  + Parameter: const PORT = 3000
* **Database Path:** The database connection logic in database.js automatically handles the path. For development (npm start), it uses a local clinic.db file in the project root. For the packaged application, it creates the database in the user's application data directory (%APPDATA%).
* **Default Admin Account:** The initial administrator account is created on the first run of the server if no users exist.
  + File: app.js
  + Function: initAccounts()
  + Parameters: const username = "admin", const password = "123"

**CHAPTER V**

**API DOCUMENTATION**

The backend exposes the following RESTful API endpoints.

**Authentication (/api/auth)**

* **POST /login**
  + **Description:** Authenticates a user.
  + **Request Body:** {"username": "admin", "password": "123"}
  + **Success Response (200):** {"success": true, "data": "Login successful!"}
  + **Error Response (401):** {"success": false, "data": "Invalid username or password."}

**Patient Management (/api/patients)**

* **GET /**: Retrieves a list of all patients.
* **POST /**: Creates a new patient.
  + **Request Body:** {"name": "John Doe", "dob": "1990-01-15", "address": "123 Main St"}
* **GET /:id**: Retrieves a single patient by their ID.
* **PUT /:id**: Updates a patient's information.
* **DELETE /:id**: Deletes a patient and their associated consultations.
* **GET /search?name=:query**: Searches for patients by name.

**Consultation Management (/api/patients/:patient\_id/consultations)**

* **GET /**: Retrieves all consultations for a specific patient.
* **POST /**: Adds a new consultation for a specific patient.
  + **Request Body:** {"complaint": "Headache", "diagnosis": "Migraine", "treatment": "Rest", "consultation\_date": "2025-09-30"}

**Medicine Inventory (/api/medicines)**

* **GET /**: Retrieves a list of all medicines.
* **POST /**: Creates a new medicine.
  + **Request Body:** {"name": "Paracetamol", "quantity": 100, "description": "500mg tablets"}
* **GET /search?name=:query**: Searches for medicines by name.
* **PUT /:id**: Updates a medicine's quantity and/or description.
* **DELETE /:id**: Deletes a medicine from the inventory.

**CHAPTER VI**

**DATABASE DOCUMENTATION**

A screenshot of a computer screen

AI-generated content may be incorrect.

**Table Schema**

Refer to the Design Document for a detailed breakdown of each table, its columns, data types, and constraints.

**Data Backup & Migration**

The database is a single file (clinic.db). Backup is a manual process of copying this file from its storage location (%APPDATA%/CRMS Thesis/clinic.db) to a secure external drive. There is no automated data migration process; initial data must be entered manually.

**CHAPTER VII**

**USER MANUAL**

Refer to the separate **User Manual** document for detailed, non-technical instructions on how to use the CRMS application. Core workflows include user login, patient CRUD operations, consultation management, and inventory tracking.

**CHAPTER VIII**

**TROUBLESHOOTING GUIDE**

Issue: Application shows a blank white screen.

* Cause: A critical JavaScript error on the frontend.
* Resolution: Press Ctrl+Shift+I to open the Developer Tools. Check the "Console" tab for red error messages to diagnose the problem.

Issue: Data is not saving, or an error modal appears.

* Cause: A backend server error or network issue.
* Resolution: If running in development (npm start), check the terminal for any server-side error logs. These often indicate database constraint violations or other logic errors.

Issue: Application fails to launch.

* Cause: Corrupted installation or missing dependencies.
* Resolution: Try reinstalling the application using the .exe installer. For developers, run npm install to ensure all packages are present.

**CHAPTER IX**

**CODE DOCUMENTATION**

**Code Structure**

* main.js: The entry point for Electron. Creates the browser window.
* app.js: The entry point for the Node.js/Express.js backend server.
* database.js: Manages SQLite database connection and table initialization.
* /public: Contains all frontend assets (HTML, CSS, JavaScript files).
* /controllers: Contains the business logic for each API resource.
* /routes: Defines the API endpoints and links them to controller functions.

**Inline Comments**

Key functions and complex logic blocks within the controllers are commented to explain their purpose.

**Coding Standards**

The project follows standard JavaScript ES6+ conventions, including the use of async/await for asynchronous operations. Code is formatted using standard practices to ensure readability.

**CHAPTER X**

**TESTING DOCUMENTATION**

**CHAPTER XI**

**MAINTENANCE GUIDE**

**CHAPTER XII**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes |
| 1.0 | 09/08/2025 | Catubay, Mark Lawrence | Initial draft of the Technical Documentation. |

**CHAPTER XIII**

**APPROVAL**

**CHAPTER XIV**

**APPENDIX**