Clinic Management System - Detailed Project Report

1. Introduction & Overview

1.1 Project Title & Abstract

Project Title: Clinic Management System

Abstract:

Clinic Management System is a web-based healthcare workflow platform designed to improve coordination between doctors and receptionists, while supporting patient appointment and record management. The application is implemented using the MERN stack: MongoDB, Express.js, React, and Node.js. It digitizes patient registration, token/queue handling, doctor consultation updates, prescription capture, and billing operations in a centralized system.

1.2 Problem Statement

Traditional paper-based or partially manual clinic systems commonly face:

- Data redundancy and duplicate records
- Delayed access to patient history
- Limited visibility across receptionist and doctor workflows
- Poor traceability for prescriptions and billing steps
- Increased risk of human errors in queue and payment tracking

This project addresses these issues by moving operational workflows to structured digital modules backed by a centralized database.

1.3 Objectives

- Digitize patient profile and visit records
- Automate queue token generation and status transitions
- Improve doctor-receptionist coordination through shared workflow states
- Maintain medical history and prescriptions for future reference
- Support billing lifecycle including pending transactions and payment completion
- Improve clinic-level operational efficiency and reporting readiness

1.4 Project Scope

In Scope:

- Role-based authentication and authorization
- Doctor/receptionist workflows
- Patient registration and appointment booking
- Walk-in and scheduled queue operations
- Consultation records (diagnosis, symptoms, prescription)
- Billing and invoice status management

- Documentation and test artifacts for submission

Out of Scope:

- Insurance claims integration
- Full e-pharmacy integration
- PACS-grade medical imaging platform
- External payment gateway settlement reconciliation (core status flow is implemented)

2. System Requirements (SRS)

2.1 Functional Requirements

- Patient can register and maintain profile data
- Patient can view doctors and book appointments
- Receptionist can register walk-in patients
- Receptionist can generate queue tokens automatically
- Receptionist can move scheduled patients to queue
- Doctor can view assigned appointments and queue entries
- Doctor can record diagnosis, symptoms, and prescriptions
- Receptionist can generate, edit, and process bills
- System can retain patient history for future retrieval
- System can expose role-based dashboards and module actions

2.2 Non-Functional Requirements

- **Security**: JWT-based authentication, password hashing, role access middleware
- **Scalability**: Layered backend architecture with modular controllers/services/models
- **Reliability**: Structured error handling and validation middleware
- **Performance**: Indexed document queries and efficient state filtering
- **Usability**: Responsive dashboard-based UI with module-specific navigation
- **Maintainability**: Standardized folder structure and coding standards documentation
- **Portability**: Runs on Windows/Linux/macOS with environment-based configuration

2.3 Hardware & Software Specifications

Minimum Hardware:

- CPU: Dual-core processor (Intel i3/Ryzen 3 or equivalent)
- RAM: 4 GB minimum (8 GB recommended)
- Storage: 2 GB free for dependencies, logs, and local builds

Software / Development Environment:

- OS: Windows 10/11, Linux, or macOS

- Editor: Visual Studio Code
- Runtime: Node.js (v16+)
- Package Manager: npm
- Backend: Express.js
- Frontend: React + Vite
- Database: MongoDB (local/Atlas)
- API Testing: Postman or curl
- Testing: Jest + Supertest

3. System Design & Architecture

3.1 Three-Tier Architecture

- **Client Tier (React)**: UI, forms, routing, dashboard views, API consumption
- **Application Tier (Node.js/Express)**: Business logic, validation, authentication, module orchestration
- **Data Tier (MongoDB)**: Persistent document storage for users, appointments, queue, medical records, and billing

Request flow:

`React UI -> REST API (Express Controllers/Services) -> MongoDB -> Response -> UI update`

3.2 Diagrams

3.2.1 Use Case Diagram (Textual)

Actors:

- Admin/Receptionist
- Doctor
- Patient

Core interactions:

- Patient: register/login, book appointment, view records/billing
- Receptionist: register walk-in, generate token, manage queue, process billing
- Doctor: view appointments/queue, update diagnosis/prescription, complete consultation

3.2.2 Data Flow Diagram (DFD) - Level 1 (Textual)

- Input processes:
- Registration/Login
- Appointment Booking
- Walk-in Registration and Token Creation
- Consultation Record Update
- Billing and Payment Update

- Data stores:
- User Store
- Appointment Store
- Queue Store
- Medical Record Store
- Billing Store
- Outputs:
- Dashboard status updates
- Queue position/status updates
- Prescriptions and billing data retrieval

3.2.3 ER Diagram (Entity Relationships) - Textual

- `User (1) -> (0..1) Doctor`
- `User (1) -> (0..1) Patient`
- `Patient (1) -> (N) Appointment`
- `Doctor (1) -> (N) Appointment`
- `Appointment (1) -> (0..1) Queue`
- `Appointment (1) -> (0..1) Billing`
- `Patient (1) -> (N) MedicalRecord`
- `Doctor (1) -> (N) MedicalRecord`

3.3 Schema Design (Mongoose Models)

Primary collections:

- `User`
- `Doctor`
- `Patient`
- `Appointment`
- `Queue`
- `MedicalRecord`
- `Billing`
- `Service`
- `Department`
- `Feedback`
- `MedicalDocument`

Model definitions and validation rules are implemented in `backend/models/`.

4. Implementation & Modules

4.1 Project Structure

- `frontend/`: React UI, routing, API clients, pages, shared components, Redux slices
- `backend/`: Express server, controllers, services, models, middleware, routes, tests, config
- Root docs: architecture, LLD, testing, optimization, compliance, wireframe, function reference

4.2 Key Modules

4.2.1 Authentication

- JWT token flow with protected routes
- Role-aware authorization middleware for doctor/receptionist/patient modules
- Password hashing and secure login processing

4.2.2 Patient Management

- Patient registration and profile storage
- Walk-in registration workflow
- Token generation and queue linkage
- Medical history and visit record persistence

4.2.3 Appointment Module

- Appointment booking for patients
- Status transitions (scheduled, arrived, in-consultation, pending-billing, completed)
- Doctor/receptionist synchronized views
- Date/time and doctor-level scheduling flow

4.2.4 Billing & Invoicing

- Bill creation from consultation context
- Automatic consultation fee support from doctor profile
- Additional charges (tests/x-ray/etc.), tax/discount, totals
- Payment processing and queue status closure handling

5. Testing & Screenshots

5.1 System Testing

Testing coverage includes:

- Unit-level controller/service behavior checks
- Integration-oriented module workflow checks
- Role-based access and action path validation

Validation status observed:

- Backend tests: 57 passed

- Backend lint: pass
- Frontend lint: pass
- Frontend production build: pass

5.2 UI Screenshots

Include screenshots in final submission for:

- Login page(s)
- Receptionist dashboard
- Walk-in queue/token module
- Doctor appointments and consultation form
- Billing edit/process workflow
- Patient booking screen

Suggested screenshot folder:

- `submission-assets/screenshots/`
- `login.png`
- `receptionist-dashboard.png`
- `walkin-queue.png`
- `doctor-appointments.png`
- `billing-process.png`
- `patient-book-appointment.png`

6. Conclusion & Future Work

6.1 Conclusion

The Clinic Management System successfully digitizes key clinic operations and reduces dependency on manual coordination. The implementation improves visibility between doctor and receptionist workflows, standardizes queue and billing status transitions, and preserves patient history for longitudinal clinical use.

6.2 Future Scope

- Payment gateway integration for online transactions
- SMS/WhatsApp notification automation for appointment/token updates
- Diagnostic image and report lifecycle extension (advanced storage and viewer)
- Audit dashboard for operational KPIs and compliance reporting
- Advanced analytics for doctor utilization and patient wait-time optimization

Appendix (Optional References)

- `README.md`
- `ARCHITECTURE.md`

- `LLD.md`
- `WIREFRAME.md`
- `FUNCTION_REFERENCE.md`
- `TESTING.md`
- `OPTIMIZATION.md`
- `REQUIREMENTS_COMPLIANCE.md`