Project Title : Medicine Tracker

Submitted by:
Aditya Chaudhary

Supervisor:
Anuj Kumar

Program:
Web Development using React (WDR)

Department:
Department of Computer Science / Information Technology

Institution: Dr. K.N. Modi Institute of Engineering and Technology

Degree:
Bachelor of Technology (B.Tech)

Submission Date: 07/01/2026

Abstract

Medication adherence plays a critical role in effective healthcare delivery, yet a significant number of patients fail to take prescribed medicines correctly due to forgetfulness, complex schedules, or lack of monitoring tools. The **Medicine Tracker** application is designed to address this problem by providing a digital platform for managing medication schedules, reminders, dosage logs, and stock levels in an integrated manner.

The proposed system enables users to register securely, add medicines with detailed dosage schedules, receive automated reminders, and log medicine intake. Additionally, the system tracks remaining stock quantities and alerts users when medicines are running low. The backend of the application is developed using a RESTful architecture, incorporating secure authentication mechanisms and background services for reminder scheduling. The frontend is implemented using a modern JavaScript framework to ensure a responsive, accessible, and intuitive user experience across devices.

This project emphasizes modular design, separation of concerns, and scalability. Key results include reliable reminder delivery, accurate adherence tracking, and improved user engagement with medication routines. The Medicine Tracker application demonstrates how full-stack web technologies can be effectively applied to solve real-world healthcare problems while maintaining security, usability, and performance. The system can be extended in the future to include analytics, predictive adherence monitoring, and integration with external healthcare services.

Acknowledgements

The author expresses sincere gratitude to the project supervisor for their guidance, encouragement, and valuable feedback throughout the development of this project. Special thanks are extended to the faculty members of the Department of Computer Science/Information Technology for providing the foundational knowledge required to undertake this work. Appreciation is also conveyed to peers and family members for their continuous support and motivation during the project lifecycle.

Table of Contents

Chapter 1: Introduction

1.1 Background and Motivation

Medication adherence is a persistent challenge in modern healthcare systems. Studies indicate that a substantial percentage of patients fail to follow prescribed medication regimens, leading to treatment inefficacy, increased hospital admissions, and avoidable healthcare costs. Factors contributing to non-adherence include forgetfulness, complex dosing schedules, lack of reminders, and insufficient tracking mechanisms.

With the rapid adoption of smartphones and web technologies, digital health solutions have emerged as effective tools for improving patient engagement and self-management. However, many existing applications lack comprehensive features such as stock monitoring, detailed dosage logging, or secure, scalable architectures. This gap motivates the development of the **Medicine Tracker** application.

1.2 Overview of the Medicine Tracker System

The Medicine Tracker system is designed as a full-stack web application that enables users to:

- Manage medicines and dosage schedules
- Receive timely reminders
- Log medication intake
- Monitor medicine stock levels
- View adherence history

The system targets individual users, caregivers, and can be extended to healthcare providers. It emphasizes usability, security, and extensibility.

1.3 Value Proposition

The Medicine Tracker offers:

- Improved medication adherence
- Reduced risk of missed or incorrect dosages

- Better visibility into medicine usage
- A scalable foundation for future healthcare integrations

1.4 Glossary of Key Terms

- **Medication Adherence:** Degree to which a patient follows prescribed medication regimens
- **REST API:** Representational State Transfer Application Programming Interface
- **Cron Job:** Scheduled background task
- **JWT:** JSON Web Token used for authentication

Chapter 2: Problem Statement and Objectives

2.1 Problem Statement

Patients often struggle to manage multiple medicines with varying dosage schedules. Traditional reminder methods such as alarms or written notes are unreliable and lack tracking capabilities. Existing digital solutions frequently fail to provide an integrated, secure, and extensible platform.

2.2 Objectives

2.2.1 Primary Objectives

- To design and develop a secure medicine tracking system
- To automate medication reminders
- To log dosage adherence accurately

2.2.2 Secondary Objectives

- To monitor medicine stock levels
- To provide a responsive and accessible user interface
- To ensure scalability and maintainability

2.3 Success Criteria

- Accurate reminder delivery
- Secure user authentication
- Positive usability feedback
- Reliable data storage and retrieval

Chapter 3: Literature Review

3.1 Overview of Medication Management Systems

Medication management systems have been extensively studied in healthcare informatics. Digital reminder systems have been shown to improve adherence by up to 20–30% in chronic disease management scenarios [1].

3.2 Review of Existing Applications

Popular applications such as Medisafe and MyTherapy provide reminder functionalities but often lack extensibility, detailed stock management, or open system architectures.

3.3 Academic Research on Adherence Monitoring

Research emphasizes the importance of personalized reminders, logging mechanisms, and feedback loops. Systems integrating reminder automation with adherence analytics show improved outcomes [2].

3.4 Identified Gaps

- Limited customization of dosage schedules
- Lack of backend extensibility
- Insufficient focus on stock monitoring

3.5 Design Decisions Informed by Literature

Based on the literature, the Medicine Tracker system incorporates:

- Automated reminders via background services
- Comprehensive logging mechanisms
- Modular, scalable architecture

Chapter 4: System Analysis

4.1 Stakeholder Analysis

A stakeholder is any individual or entity that interacts with or is affected by the system. Identifying stakeholders helps define requirements and system boundaries.

4.1.1 Primary Stakeholders

- **End Users (Patients):**
  Individuals who use the system to manage their medication schedules, receive reminders, and track dosage adherence.
- **Caregivers:**
  Family members or caretakers who assist patients in monitoring medication intake.
- **System Administrator:**
  Responsible for system maintenance, monitoring logs, and managing configurations.

4.1.2 Secondary Stakeholders

- **Healthcare Professionals (Future Extension):**
  Doctors or pharmacists who may review adherence reports.
- **Developers and Maintainers:**
  Responsible for system enhancement and bug fixes.

4.2 Use Case Analysis

Use cases describe interactions between users and the system.

4.2.1 Primary Use Cases

- User Registration and Login
- Add / Update / Delete Medicine
- Schedule Dosage
- Receive Reminder Notification
- Log Medicine Intake

- Monitor Medicine Stock
- View Adherence History

4.2.2 Use Case Example: Add Medicine

Actor: User
Description: User adds a new medicine with dosage schedule.

Main Flow:

1. User logs into the system.
2. Navigates to “Add Medicine”.
3. Enters medicine name, dosage, frequency, and stock quantity.
4. Submits the form.
5. System validates data and stores it in the database.

Alternate Flow:

- If invalid data is entered, system displays validation errors.

4.3 User Stories

User stories describe functionality from the user’s perspective.

- *As a user, I want to receive reminders so that I do not forget to take my medicine.*
- *As a user, I want to track remaining medicine stock to avoid running out unexpectedly.*
- *As a caregiver, I want to see adherence history to ensure medicines are taken on time.*

4.4 Functional Requirements

Table 4.1: Functional Requirements

ID	Requirement Description
FR1	System shall allow users to register and authenticate securely
FR2	System shall allow users to add, update, and delete medicines
FR3	System shall schedule medicine reminders automatically
FR4	System shall log dosage intake
FR5	System shall monitor and alert low stock

4.5 Non-Functional Requirements

Table 4.2: Non-Functional Requirements

Category	Description
Security	Secure authentication, encrypted passwords
Performance	API response time < 500 ms
Scalability	Support increasing number of users
Availability	99% uptime
Accessibility	WCAG-compliant UI

Chapter 5: System Architecture and Design

5.1 Architectural Overview

The Medicine Tracker follows a **three-tier architecture**:

1. **Presentation Layer (Frontend)**
2. **Application Layer (Backend API)**
3. **Data Layer (Database)**

This architecture ensures separation of concerns, maintainability, and scalability.

5.2 High-Level Architecture Diagram

Figure 5.1: High-Level System Architecture

Placeholder:
Diagram showing:

- Frontend (Web App)
- Backend REST API
- Database
- Cron Job Scheduler
- Notification Service

Description:
The frontend communicates with the backend through REST APIs. The backend interacts with the database and triggers background cron jobs responsible for reminders and stock alerts.

5.3 Component Diagram

Figure 5.2: Component Diagram of Medicine Tracker

Components:

- Authentication Module
- Medicine Management Module

- Reminder Scheduler
- Notification Service
- Logging & Monitoring Module

Each component is loosely coupled and communicates via service interfaces.

5.4 Sequence Diagrams

5.4.1 Medicine Reminder Flow

Figure 5.3: Sequence Diagram – Reminder Delivery

Steps:

1. Cron job triggers scheduler.
2. Scheduler checks upcoming dosage times.
3. Notification service sends reminder.
4. User receives notification.

5.5 Data Flow and Integration

- Frontend sends JSON requests to backend APIs.
- Backend validates, processes, and persists data.
- Cron jobs periodically scan schedules.
- Notification service integrates with email/push services.

5.6 Design Principles Applied

- Modularity
- Loose Coupling
- High Cohesion
- RESTful Design
- Scalability

Chapter 6: Database Design

6.1 Choice of Database

A **Relational Database Management System (RDBMS)** is selected due to:

- Structured data requirements
- Strong consistency
- Support for relationships and constraints

Example: **PostgreSQL / MySQL**

6.2 Entity Relationship Diagram (ERD)

Figure 6.1: Entity Relationship Diagram

Placeholder:
ER diagram showing:

- users
- medicines
- dosage_schedule
- dosage_log
- notifications
- stock
- stock_log

6.3 Entity Descriptions

6.3.1 Users Table

- user_id (PK)
- name
- email
- password_hash
- created_at

6.3.2 Medicines Table

- medicine_id (PK)
- user_id (FK)
- name
- description

6.3.3 Dosage_Schedule Table

- schedule_id (PK)
- medicine_id (FK)
- dosage
- frequency
- time

6.3.4 Dosage_Log Table

- log_id (PK)
- schedule_id (FK)
- taken_at
- status

6.3.5 Stock Table

- stock_id (PK)
- medicine_id (FK)
- quantity
- threshold

6.4 Indexes and Constraints

- Index on user_id for faster lookups
- Foreign key constraints to ensure referential integrity
- Unique constraint on email

6.5 Trade-offs and Justification

Aspect	RDBMS	NoSQL
Structure	Strong	Flexible
Consistency	High	Eventual
Suitability	✓	✕

RDBMS is preferred due to transactional integrity.

Chapter 7: Backend Implementation

7.1 Backend Technology Stack and Rationale

The backend of the Medicine Tracker system is implemented using a **RESTful web service architecture**.

7.1.1 Selected Technologies

- **Runtime Environment:** Node.js
- **Framework:** Express.js
- **Database:** PostgreSQL / MySQL
- **ORM / Query Layer:** Sequelize / Prisma
- **Authentication:** JSON Web Tokens (JWT)
- **Task Scheduling:** Cron Jobs
- **Logging:** Winston / Morgan

7.1.2 Rationale

Node.js provides non-blocking I/O, making it suitable for concurrent API requests. Express.js enables rapid API development with middleware support. JWT ensures stateless and scalable authentication.

7.2 Backend Project Structure

- **Controllers:** Handle HTTP requests and responses
- **Services:** Business logic
- **Models:** Database entities
- **Routes:** API endpoints
- **Cron:** Scheduled background jobs

7.3 API Design Principles

- RESTful naming conventions
- Stateless communication
- JSON-based request/response

- Proper HTTP status codes

7.4 Key API Endpoints

Table 7.1: API Endpoint Summary

Method	Endpoint	Description
POST	/api/auth/register	User registration
POST	/api/auth/login	User login
GET	/api/medicines	Fetch medicines
POST	/api/medicines	Add medicine
POST	/api/schedule	Create dosage schedule
POST	/api/logs	Log medicine intake

7.5 Sample API Request and Response

Request:

POST /api/medicines

```
{
  "name": "Paracetamol",
  "dosage": "500mg",
  "frequency": "Twice a day"
}
```

Response:

```
{
  "status": "success",
  "medicine_id": 12
}
```

7.6 Authentication and Authorization

- JWT tokens issued upon login
- Tokens stored securely on client
- Middleware validates token for protected routes
- Role-based access extensible

7.7 Error Handling and Logging

- Centralized error middleware
- Standardized error responses
- Logs stored with timestamps and severity levels

7.8 Background Jobs and Notifications

7.8.1 Cron Jobs

- Trigger reminder checks
- Scan low-stock medicines
- Run at configurable intervals

7.8.2 Notification Service

- Email / push notification support
- Asynchronous delivery
- Retry mechanism on failure

Chapter 8: Frontend Implementation (Detailed)

8.1 Frontend Technology Stack and Rationale

8.1.1 Selected Stack

- **Framework:** React.js
- **Routing:** React Router
- **State Management:** Context API + useReducer
- **Styling:** CSS Modules / Tailwind CSS
- **Form Handling:** React Hook Form
- **HTTP Client:** Axios
- **Testing:** Jest, React Testing Library, Cypress

8.1.2 Rationale

React offers component reusability, virtual DOM efficiency, and strong ecosystem support. Context API with reducers avoids unnecessary complexity while supporting global state.

8.2 High-Level UI Architecture

Figure 8.1: Frontend Architecture Diagram

Placeholder:
Diagram illustrating:

- Pages
- Shared components
- State layer
- API service layer

8.3 Routing and Navigation

8.3.1 Public Routes

- /login
- /register

8.3.2 Protected Routes

- /dashboard
- /medicines
- /schedule
- /logs

Route guards ensure only authenticated users can access protected pages.

8.4 Component Breakdown

8.4.1 Page Components

- LoginPage
- DashboardPage
- MedicinesPage
- SchedulePage
- LogsPage

8.4.2 Reusable Components

- Navbar
- MedicineCard
- ReminderModal
- NotificationToast
- ConfirmDialog

8.5 State Management Strategy

8.5.1 Local vs Global State

State Type	Storage
Form state	Local component
User session	Global Context

State Type	Storage
Medicine list	Global Context

8.5.2 Context API Flow

- Global store using useReducer
- Actions dispatched for CRUD operations
- Optimistic UI updates applied

8.6 Form Design and Validation

- Client-side validation using React Hook Form
- Server-side validation feedback displayed
- Error messages user-friendly and accessible

8.7 Accessibility and Responsive Design

- WCAG 2.1 compliance
- Semantic HTML elements
- Keyboard navigation
- ARIA labels for screen readers
- Mobile-first responsive layout

8.8 Styling and Theming

- Modular CSS to avoid conflicts
- Consistent spacing and typography
- Light and dark theme extensible
- Responsive grid system

8.9 API Integration Strategy

- Axios interceptors for token injection
- Centralized error handling
- Retry logic for network failures

- Loading and empty states handled gracefully

8.10 Frontend Testing Strategy

8.10.1 Unit Testing

- Component rendering tests
- Reducer logic tests

8.10.2 Integration Testing

- API integration tests

8.10.3 End-to-End Testing

- Cypress-based flow tests
- Login → Add medicine → Log intake

8.11 Performance Optimizations

- Code splitting using React.lazy
- Memoization of heavy components
- Debounced API calls
- Image and asset optimization