

Medicure winforms app

Documentation



April 15, 2025

Table of Contents

[Medicure App Documentation 2](#_Toc195657039)

[Project Overview 2](#_Toc195657040)

[Objectives 2](#_Toc195657041)

[Development Strategies 2](#_Toc195657042)

[Modular Design 2](#_Toc195657043)

[Role-Based Workflow 2](#_Toc195657044)

[File-Based Data Persistence 2](#_Toc195657045)

[Iterative and Incremental Development 3](#_Toc195657046)

[Defensive Programming & Error Handling 3](#_Toc195657047)

[Manual Testing + Automated Unit Tests 3](#_Toc195657048)

[Reusability & Scalability 3](#_Toc195657049)

[Maintainability Considerations 3](#_Toc195657050)

[Diagrams 4](#_Toc195657051)

[Hardware Configuration 4](#_Toc195657052)

[Software Configuration 5](#_Toc195657053)

[Virtual Machine Usage 5](#_Toc195657054)

[Tool Versions 5](#_Toc195657055)

[Test Results and Screenshots 5](#_Toc195657056)

[Comparisons of Results / Tools 8](#_Toc195657057)

[Interpretations 8](#_Toc195657058)

[Winforms UI Images 9](#_Toc195657059)

[Bibliographic References 14](#_Toc195657060)

# Medicure App Documentation

## Project Overview

The **Medicure App** is a comprehensive desktop-based healthcare management system designed to streamline and digitize the core operations of medical clinics and hospitals. Developed using Windows Forms, the application aims to bridge the gap between healthcare providers and administrative efficiency by providing a seamless experience in managing patients, appointments, users (such as doctors, admins), and medical records.

## Objectives

* **Digitize administrative tasks** such as patient registration, doctor management, appointment scheduling, and report generation.
* **Ensure data persistence** using flat file storage, providing a lightweight solution for clinics without database integration.
* **Improve accessibility** to patient data for healthcare providers.
* **Offer role-based access** for users (e.g., doctor vs admin).
* **Simplify report tracking** through automated data entry and retrieval.
* **Facilitate testing and quality assurance** using unit tests and robust error handling.

## Development Strategies

The development of the Medicure App followed a modular, testable, and iterative approach to ensure functionality, maintainability, and ease of enhancement. Below is a breakdown of the key strategies employed:

### Modular Design

* The application is split into well-defined classes and components:
* UserManager handles user registration and retrieval.
* DataHelper deals with patient and appointment data operations.
* Forms/UI Modules are kept separate from the logic to follow separation of concerns.
* This modular approach simplifies debugging and allows features to be extended independently.

### Role-Based Workflow

* Different user roles (e.g., Doctor, Admin) were considered from the start:
* Admins can register new users, view patients, and assign appointments.
* Doctors can only view patients and add reports. This structure enhances security and provides a tailored experience.

### File-Based Data Persistence

* Instead of using a database, the app stores all records in text files:
* users.txt, patients.txt, appointments.txt, reports.txt Each operation appends or reads structured data from these files using System.IO, reducing setup complexity and making the system portable.

### Iterative and Incremental Development

The app was built in stages:

* Basic forms and layout created.
* User and patient registration integrated.
* Appointment and reporting features added.
* Search and data retrieval implemented.
* Unit tests and validations introduced.
* Each stage included feedback and minor refactoring, ensuring continuous improvement.

### Defensive Programming & Error Handling

Robust error handling is implemented using Try-Catch blocks for file operations:

* Errors such as file not found or incorrect format are handled gracefully.
* Console logs and default returns ensure the UI remains responsive.

### Manual Testing + Automated Unit Tests

* Visual/manual testing of UI interactions during development.
* Unit tests created using testing framework to validate core functionality like:
  + User registration
  + Patient data retrieval
  + Appointment creation

### Reusability & Scalability

Functions like PatientExists, GetDoctorIdByName, and GetAppointmentsByPatientId are designed to be:

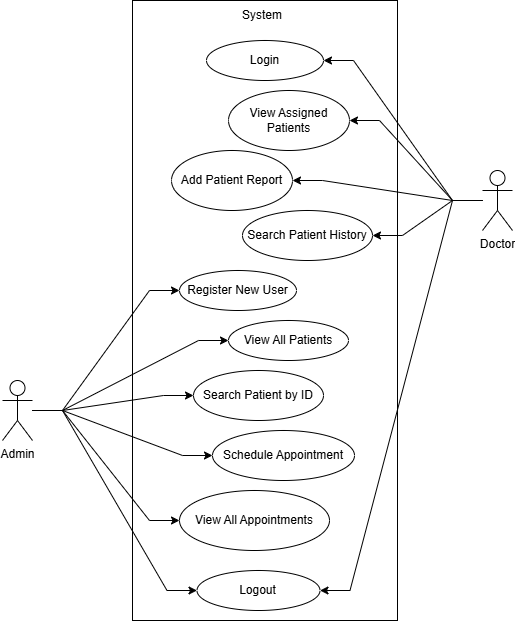
* Reusable across multiple forms.
* Easily replaceable if a switch to database storage is made in the future.

### Maintainability Considerations

* Descriptive naming for methods and variables.
* Comments added for clarity.
* Logical grouping of related methods.
* File formats are documented and easy to understand for future developers.

## Diagrams

**Use Case Diagram**



## Hardware Configuration

* CPU: Intel Core i5
* RAM: 8 GB
* Storage: 256 GB SSD
* OS: Windows 10/11

## Software Configuration

* IDE: Visual Studio 2022
* Language: VB.NET
* .NET Framework: 8.0.1
* Testing Framework: MSTest
* File Storage: Local .txt files

## Virtual Machine Usage

* **Not Used.** Application is developed and tested on a native Windows installation.

## Tool Versions

|  |  |
| --- | --- |
| Tool | Version |
| Visual Studio | 2022 (v17.x) |
| .NET Framework | 8.0.1 |
| MSTest | Latest (2025) |
| Windows OS | 10/11 |

**8. Code Snippets**

**8.1 Register User**

*Dim newUser = $"{newId},{FName},{UName},{Pass},{Role},{Gender},{Email},{MobileNo},{Salary}"*

*IO.File.AppendAllText(filePath, newUser & Environment.NewLine)*

**8.2 Get All Appointments**

*If IO.File.Exists("appointments.txt") Then*

*Dim lines = IO.File.ReadAllLines("appointments.txt")*

*For Each line In lines*

*Dim fields() = line.Split(","c)*

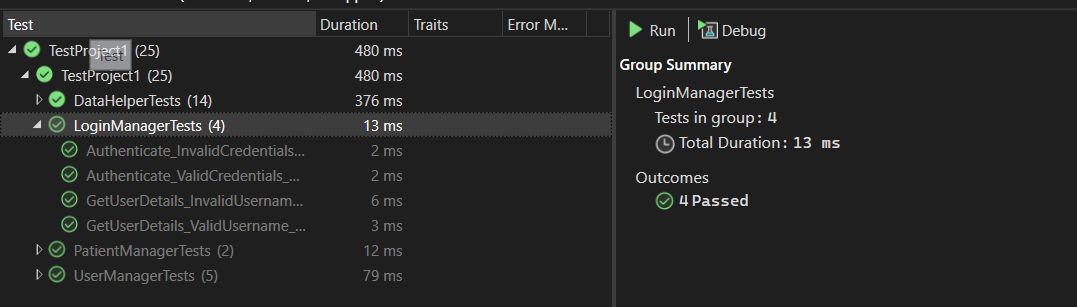
*appointmentsTable.Rows.Add(fields(0), fields(1), fields(2), fields(3), fields(4), fields(5))*

*Next*

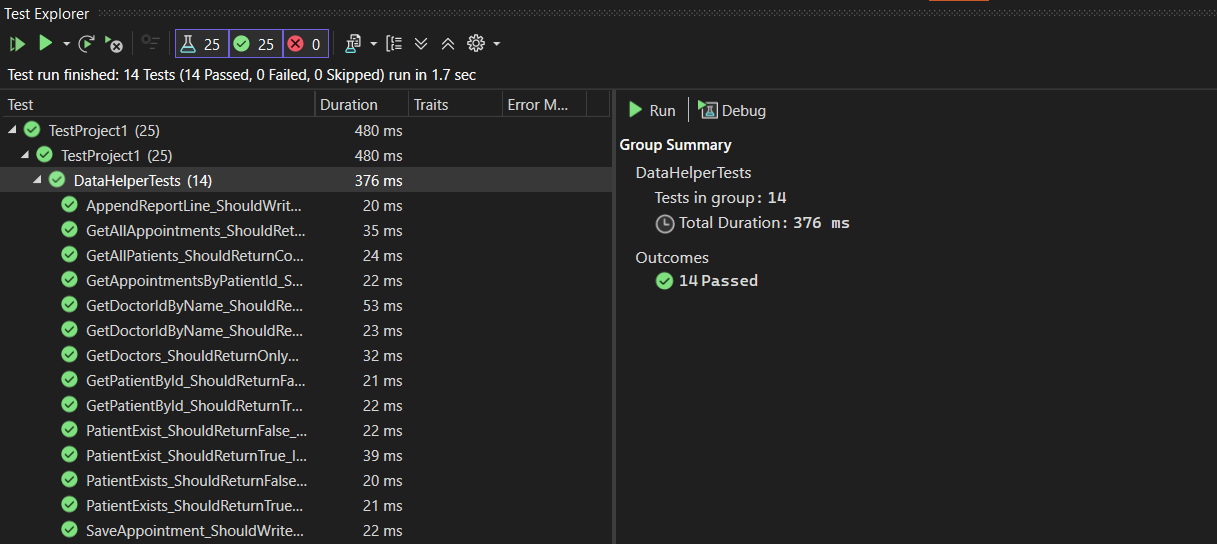
*End If*

## Test Results and Screenshots

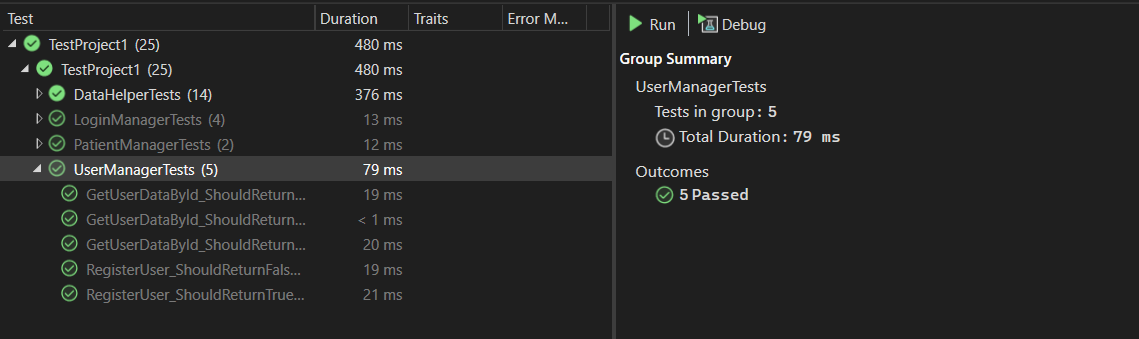
* **LoginManagerTests**
  + Authenticate\_ValidCredentials\_ReturnsTrue
  + Authenticate\_InvalidCredentials\_ReturnsFalse
  + GetUserDetails\_ValidUsername\_ReturnsDetails
  + GetUserDetails\_InvalidUsername\_ReturnsNothing



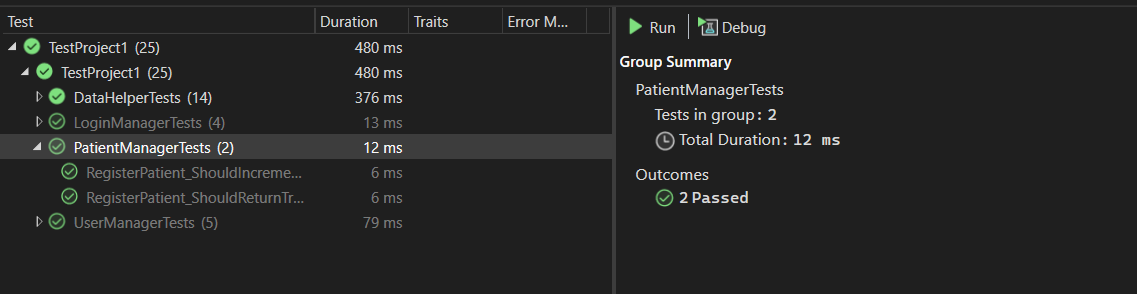
* **DataHelperTests**
  + GetPatientById\_ShouldReturnTrue\_IfPatientExists
  + GetPatientById\_ShouldReturnFalse\_IfPatientDoesNotExist
  + GetDoctors\_ShouldReturnOnlyDoctors
  + GetDoctorIdByName\_ShouldReturnCorrectId
  + GetDoctorIdByName\_ShouldReturnMinus1\_IfNotFound
  + SaveAppointment\_ShouldWriteToAppointmentsFile
  + GetAllPatients\_ShouldReturnCorrectRows
  + GetAllAppointments\_ShouldReturnExpectedRows
  + PatientExists\_ShouldReturnTrue\_IfExists
  + PatientExists\_ShouldReturnFalse\_IfNotExists
  + GetAppointmentsByPatientId\_ShouldReturnCorrectAppointments
  + PatientExist\_ShouldReturnTrue\_IfExists
  + PatientExist\_ShouldReturnFalse\_IfNotExists
  + AppendReportLine\_ShouldWriteLineToReportFile



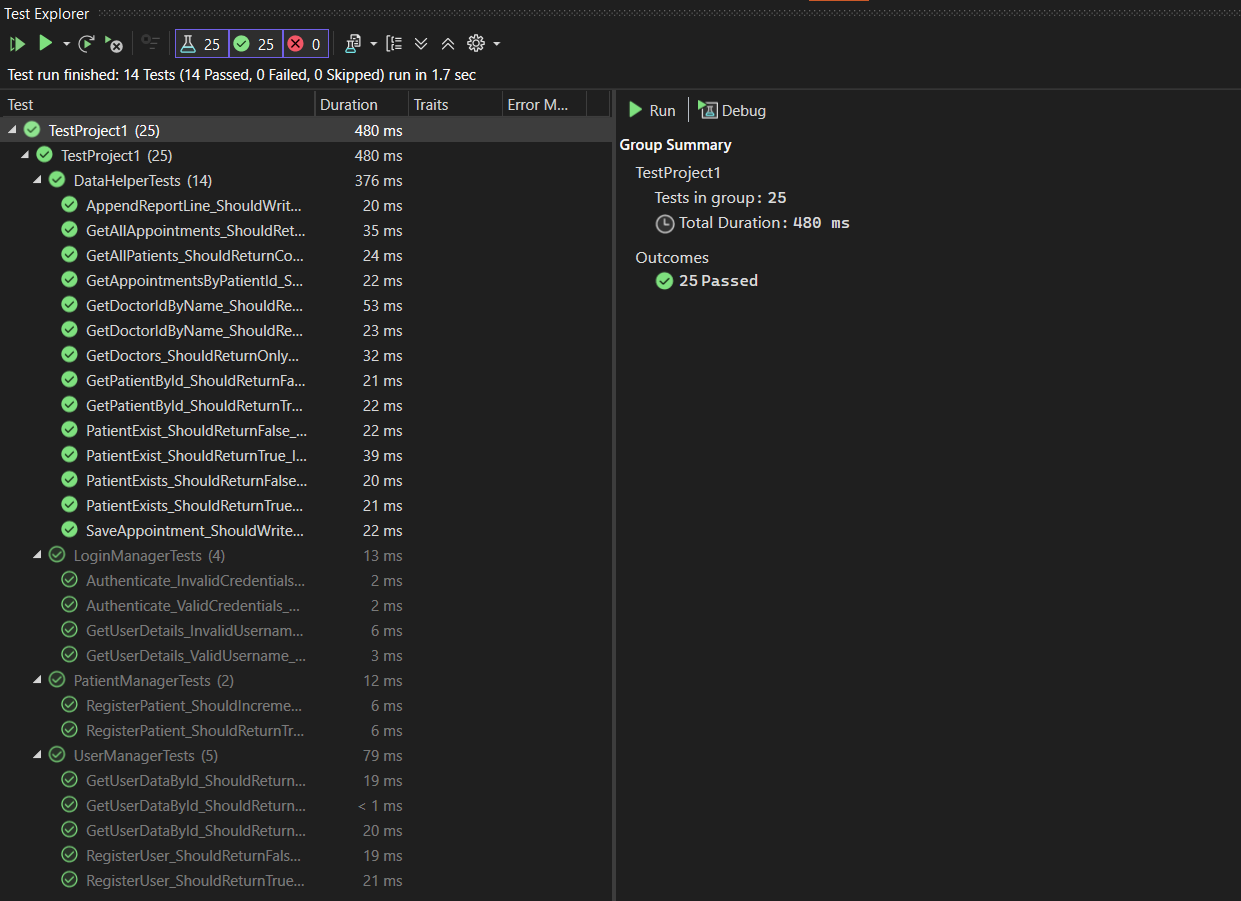
* **UserManagerTests**
  + RegisterUser\_ShouldReturnTrue\_WhenUserIsSavedSuccessfully
  + RegisterUser\_ShouldReturnFalse\_WhenFileCannotBeCreated
  + GetUserDataById\_ShouldReturnCorrectData\_WhenUserExists
  + GetUserDataById\_ShouldReturnNothing\_WhenUserDoesNotExist
  + GetUserDataById\_ShouldReturnNothing\_WhenFileDoesNotExist



* **PatientManagerTests**
  + RegisterPatient\_ShouldReturnTrue\_WhenValidDataIsProvided
  + RegisterPatient\_ShouldIncrementId\_WhenMultipleRegistrationsHappen



* **Complete Test Cases Screenshots**

****

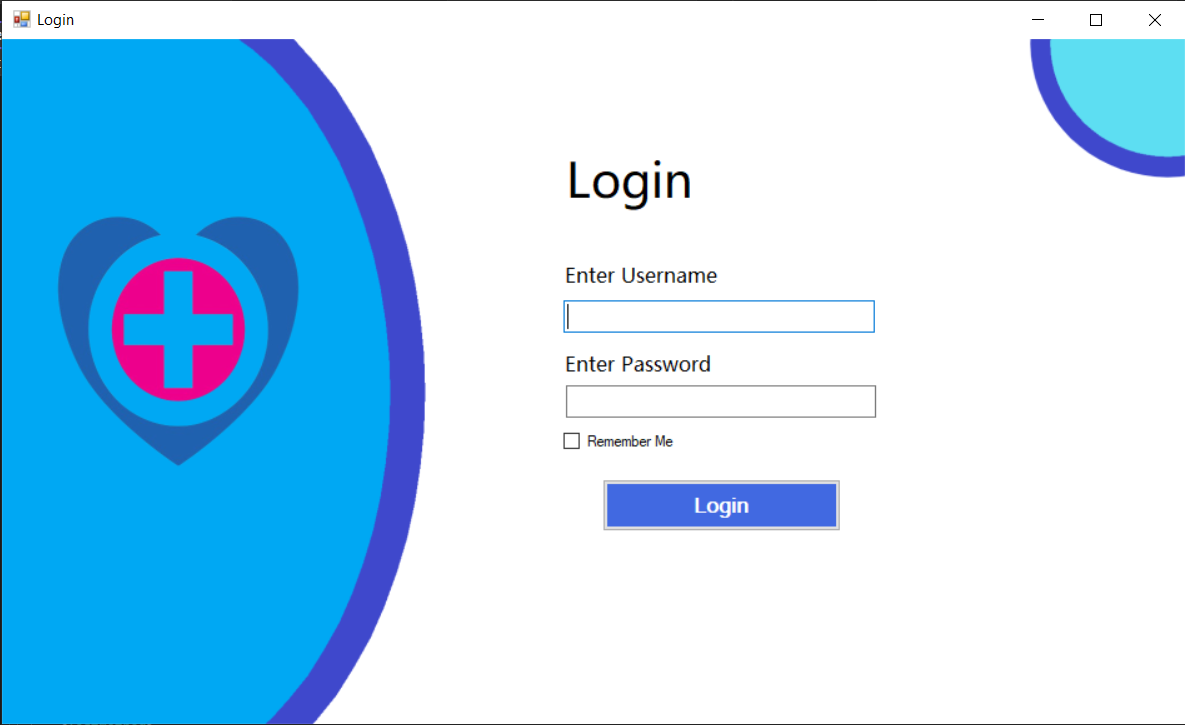
## Comparisons of Results / Tools

|  |  |  |
| --- | --- | --- |
| Feature | File-Based Storage | SQL Database |
| Setup Complexity | Low | High |
| Performance (Large Data) | Poor | Good |
| Portability | High | Medium |
| Cost | Free | May Vary |

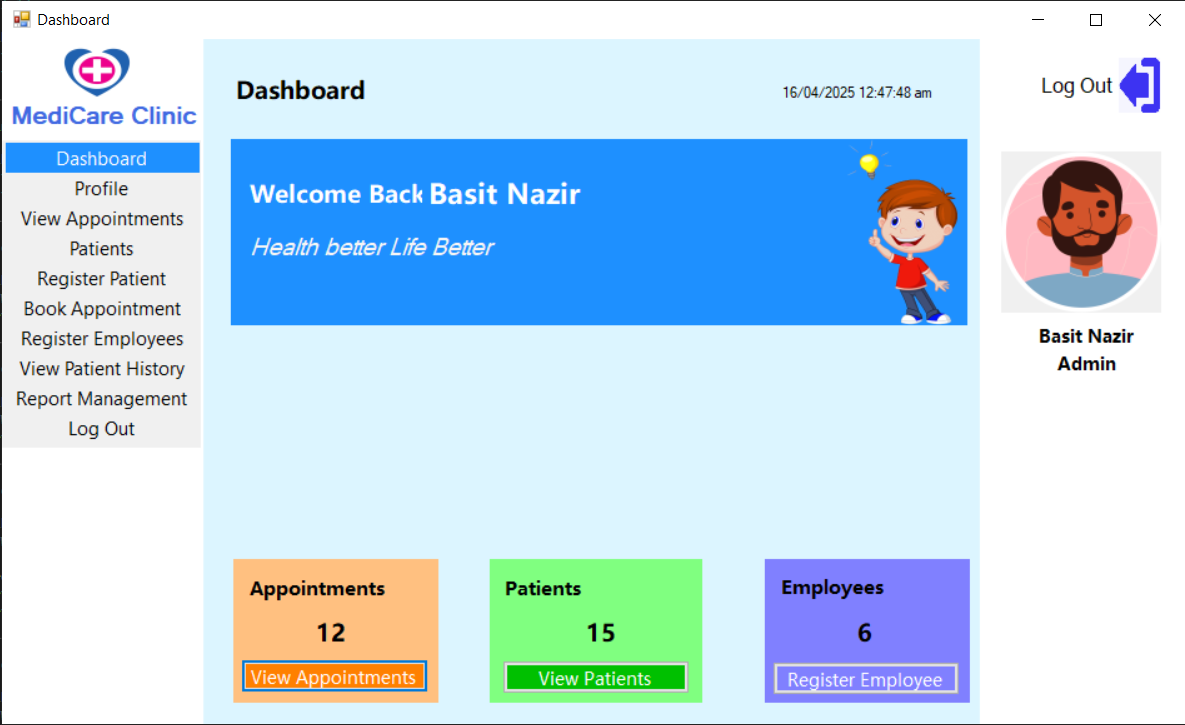
## Interpretations

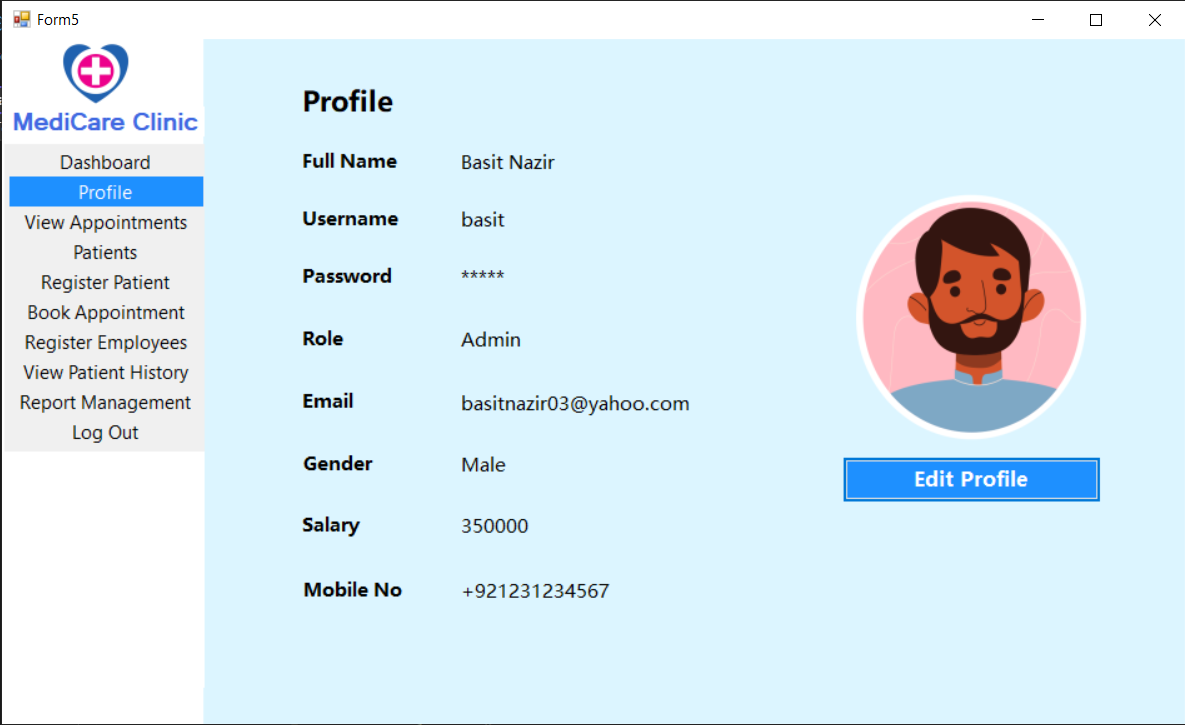
* File-based storage is quick and simple for small applications, but lacks scalability and query power.
* Using MSTest provides strong integration with Visual Studio and offers easy-to-read test results.
* The code is testable, maintainable, and suitable for small clinics without a large budget for infrastructure.

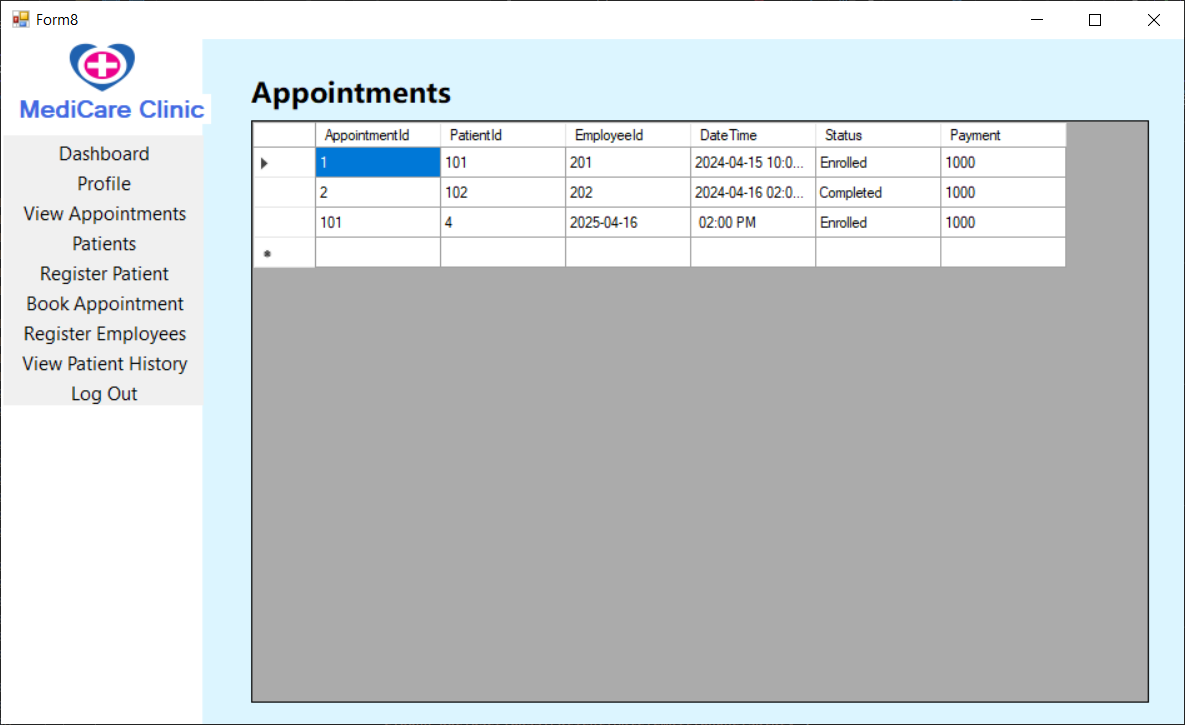
## Winforms UI Images

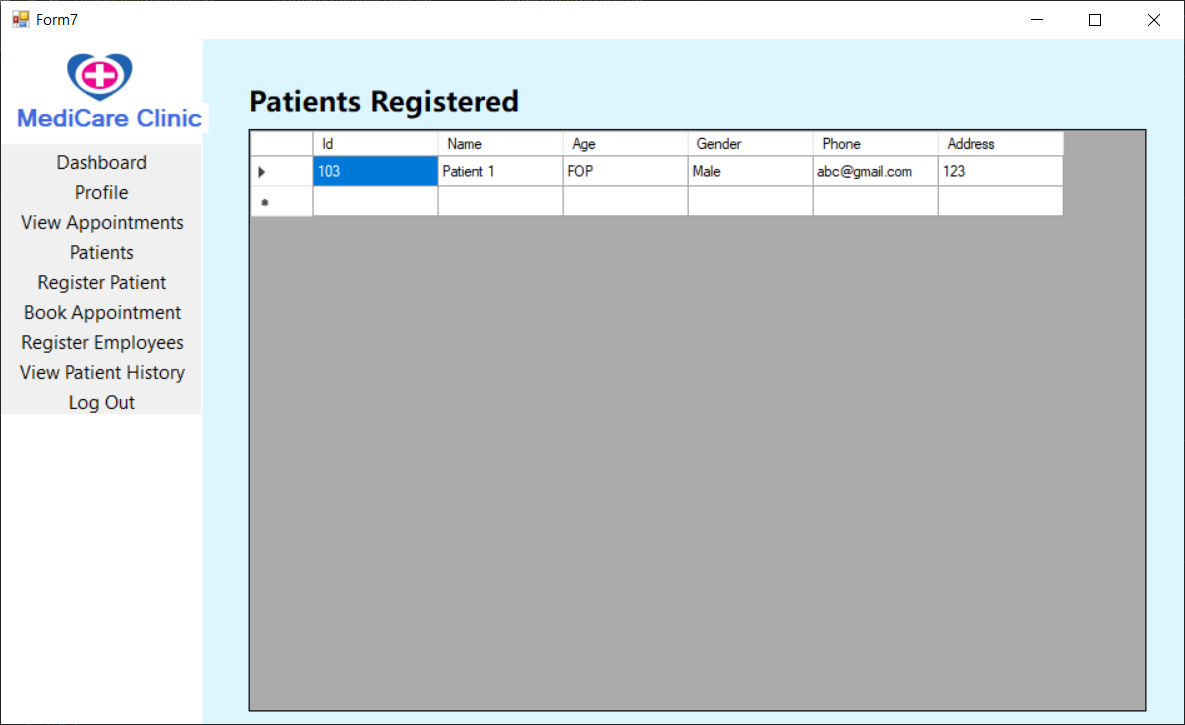




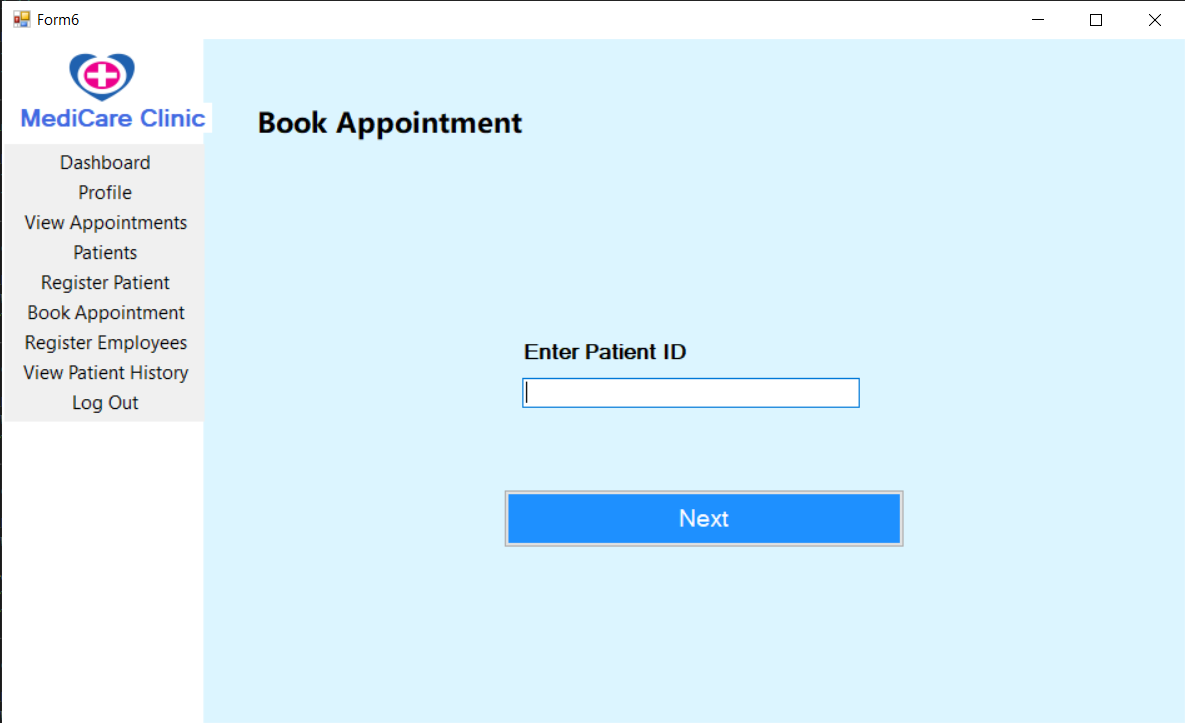


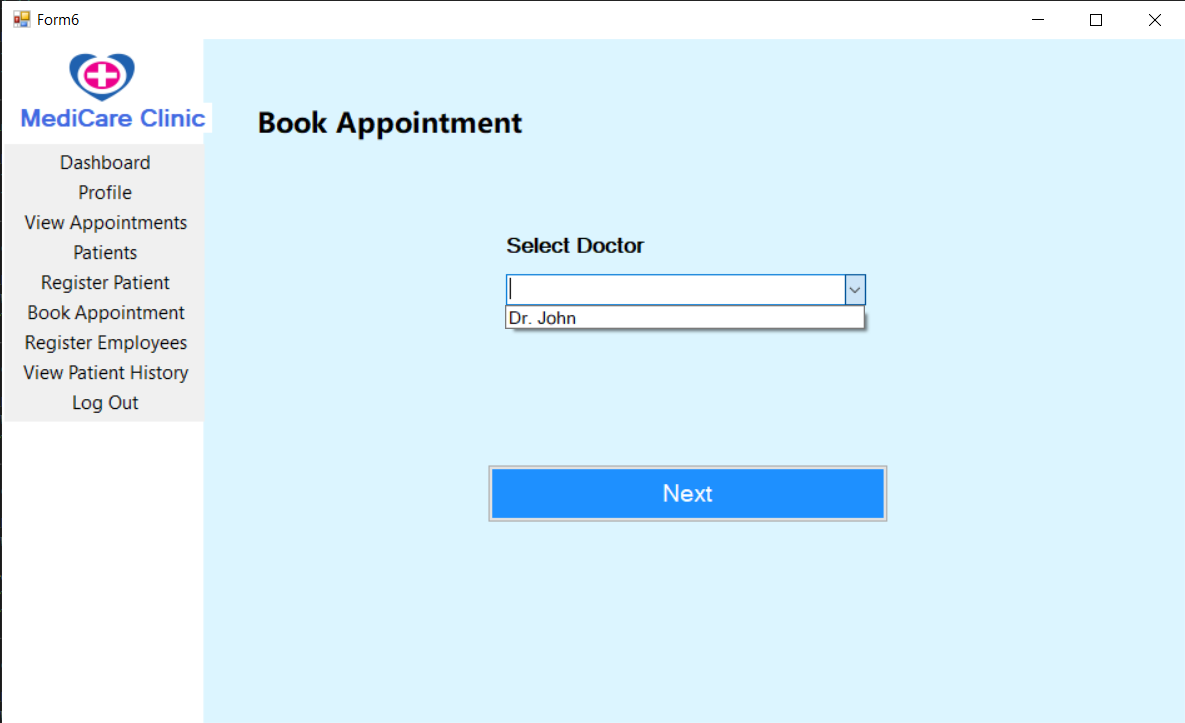


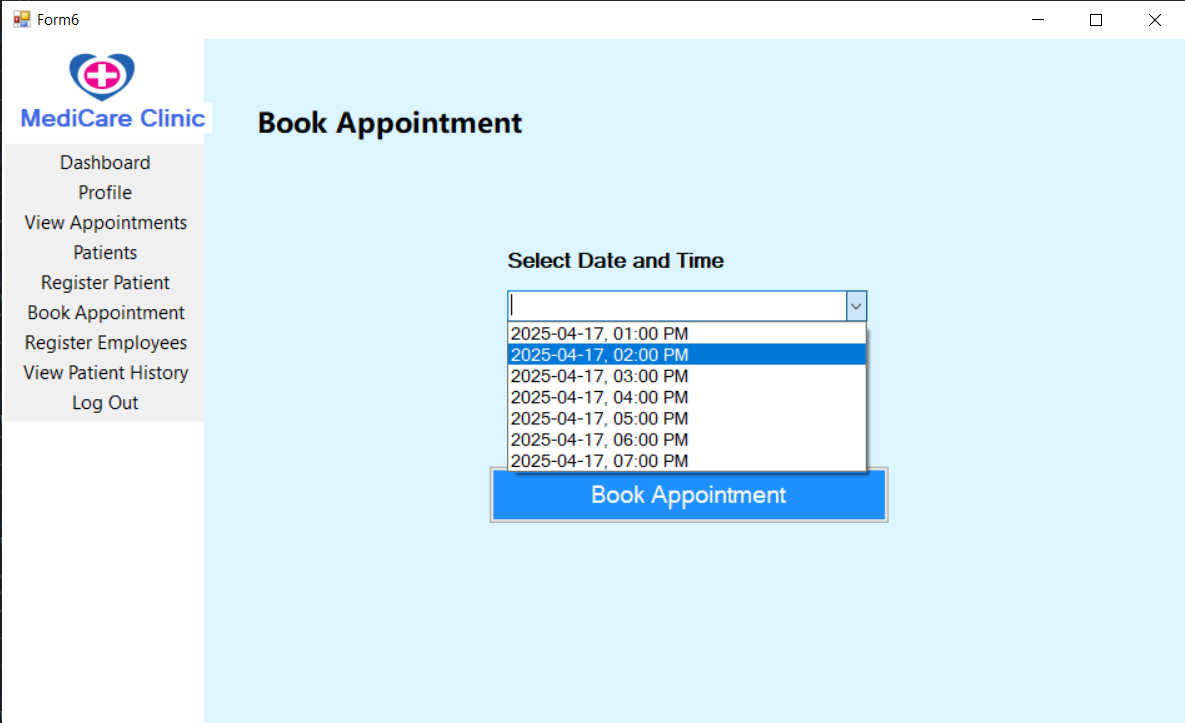


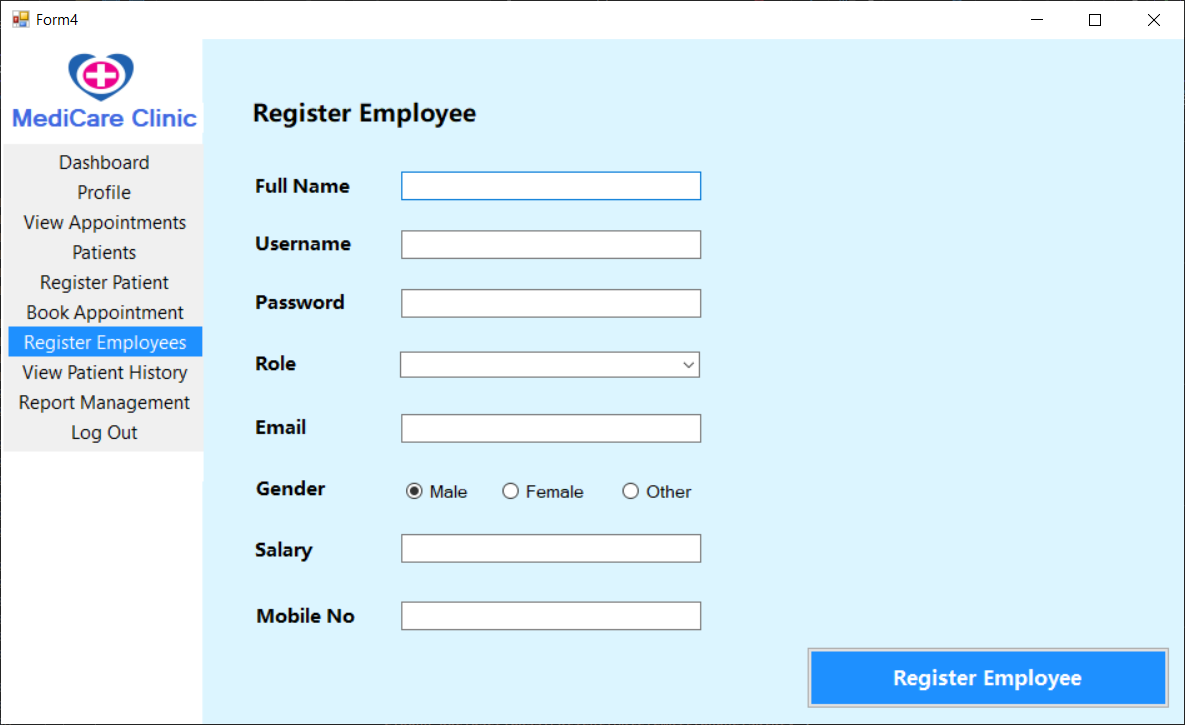


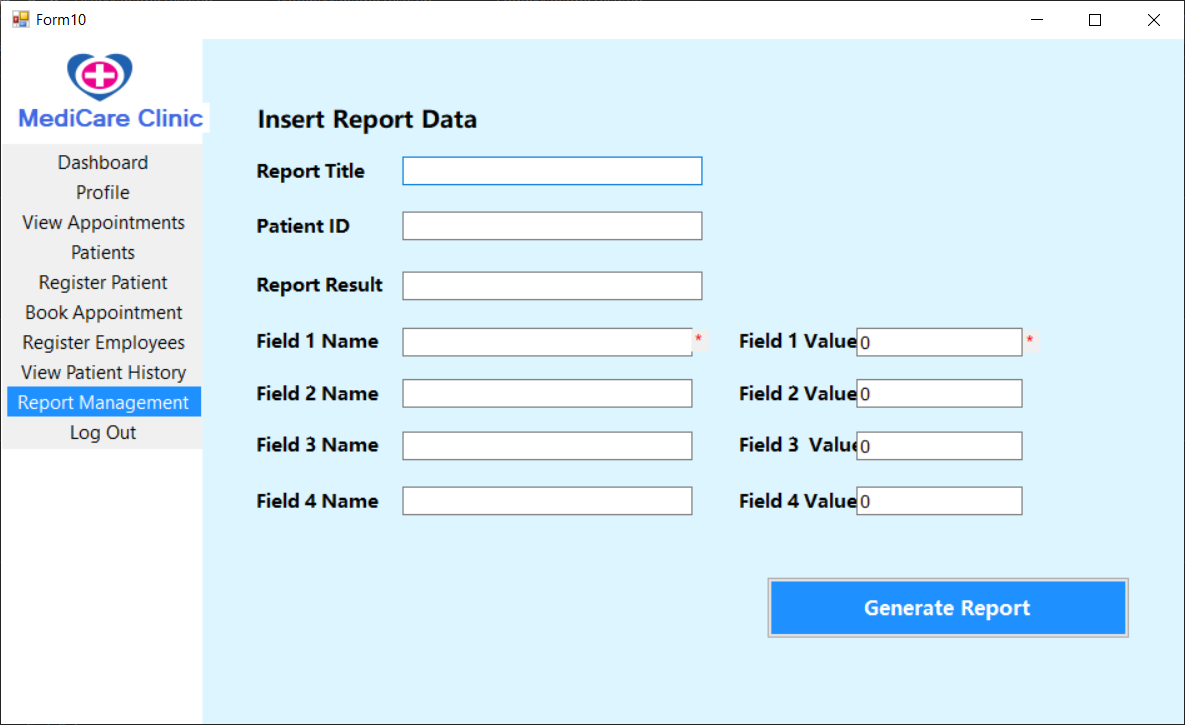












## Bibliographic References

* Microsoft Docs: <https://learn.microsoft.com/dotnet>
* MSTest: <https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest>