****

**BAHRIA UNIVERSITY, Karachi Campus**

*Department of Software Engineering*

**REPORT**

**Course Title: Software Construction Course Code**: **SEL 311**

**Course Instructor: Misbah Parveen Class**: BSE- 5(A)

**Lab Instructor:** Engr. Asma Shaheen  **Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Clinic Management System**

GROUP MEMBERS LIST:

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | Enrollment | Name | Email |
| 01. | 02-131222-124 | Sami-Ullah | M.samiiiii10@gmail.com |
| 02. | 02-131222-024 | Zuhaib Shamsher | Real.zuhaibkhan@gmail.com |
| 03. | 02-131222-008 | Owais Khalique | Owaiskhaliq0028@gmail.com |
| 04. | 02-131222-042 | Salman Ajmal | Salmankhan990@gmail.com |

**Submission Date:02/01/2025**

Contents

[**1** **INTRODUCTION** 5](#_Toc183868860)

[**1.1.** **Introduction** 5](#_Toc183868861)

[**1.2.** **Problem Statement** 5](#_Toc183868862)

[**1.3.** **Proposed Solution** 5](#_Toc183868863)

[**2** **DESIGN** 5](#_Toc183868864)

[**2.1.** **Flow Diagram** 5](#_Toc183868867)

[**2.2.** **Data Flow Diagram** 5](#_Toc183868868)

[**2.3.** **User Interfaces** 5](#_Toc183868869)

[**3** **DESCRIPTION** 5](#_Toc183868870)

[**3.1** **Technologies Use** 5](#_Toc183868871)

[**3.1.1.** **Frameworks and Platforms** 5](#_Toc183868872)

[**3.1.1.** **Programming Languages and Databases** 5](#_Toc183868873)

[**3.1.2.** **Frontend Technologies** 5](#_Toc183868874)

[**3.1.3.** **Backend Technologies** 5](#_Toc183868875)

[**3.1.4.** **Version Control and APIs** 5](#_Toc183868876)

[**3.1.5.** **Operating Environment and Production Environment** 5](#_Toc183868877)

[**3.1** **Project Perspective/Context** 5](#_Toc183868878)

[**3.2** **Project Functionalities** 5](#_Toc183868879)

[**3.3** **User Classes and Characteristics** 5](#_Toc183868880)

[**4** **REFACTORING METHODS** 5](#_Toc183868882)

[**5** **TESTING** 5](#_Toc183868883)

[**5.1** **Testing Technique** 5](#_Toc183868884)

[**5.2** **Test Cases** 5](#_Toc183868885)

[**5.3** **Test cases Implementation with outputs** 5](#_Toc183868886)

[**3.** **CONCLUSIONS AND FURTHER WORK** 5](#_Toc183868887)

[**4.** **REFERENCES** 5](#_Toc183868888)

**ABSTRACT**

The Clinic Management System (CMS) is a comprehensive software solution designed to address the operational challenges faced by modern clinics. This system streamlines processes such as patient management, appointment scheduling, and billing, ensuring efficient and secure handling of sensitive healthcare data. By leveraging cutting-edge technologies, including ASP.NET and SQL Server, the CMS provides robust features for patients, doctors, and administrators. With user-friendly interfaces, secure data management, and compliance with healthcare regulations, the CMS enhances communication, optimizes workflows, and improves patient satisfaction. This report details the system's design, functionalities, testing strategies, and future enhancements, showcasing its potential to revolutionize clinic operations.

# **INTRODUCTION**

## **Introduction**

The Clinic Management System (CMS) is an integrated solution designed to streamline clinic operations. It facilitates efficient management of patient interactions, doctor schedules, and administrative tasks while ensuring compliance with healthcare regulations. The system is web-based, making it accessible on various devices, and incorporates robust security measures to safeguard sensitive patient data.

## **Problem Statement**

Clinics face challenges such as inefficient appointment scheduling, data management, and communication between patients, doctors, and administrators. These issues can lead to operational inefficiencies, patient dissatisfaction, and errors in record-keeping.

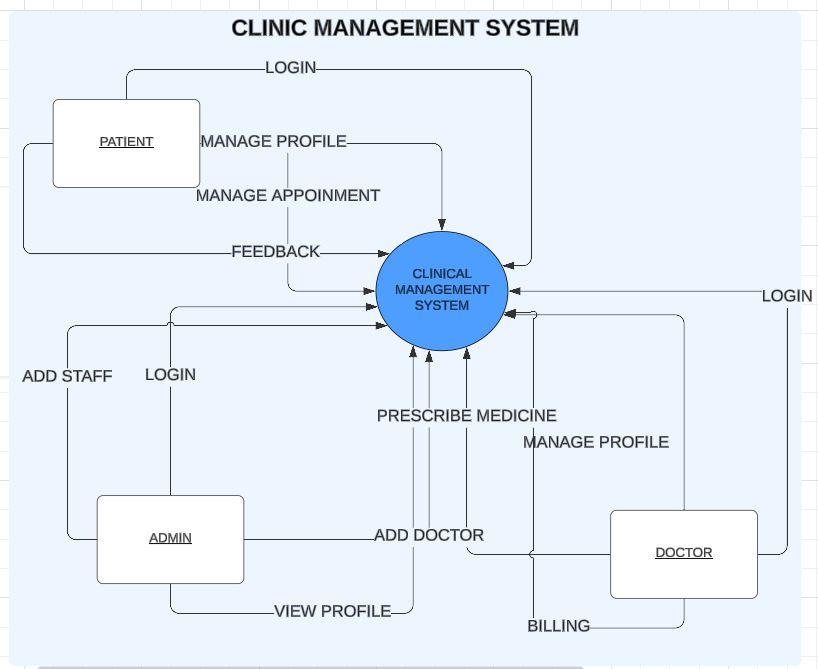
## **Proposed Solution**

The proposed CMS addresses these challenges by offering a comprehensive platform for managing appointments, billing, and patient records. It enhances communication between stakeholders, improves data accessibility, and optimizes clinic workflows through automation.

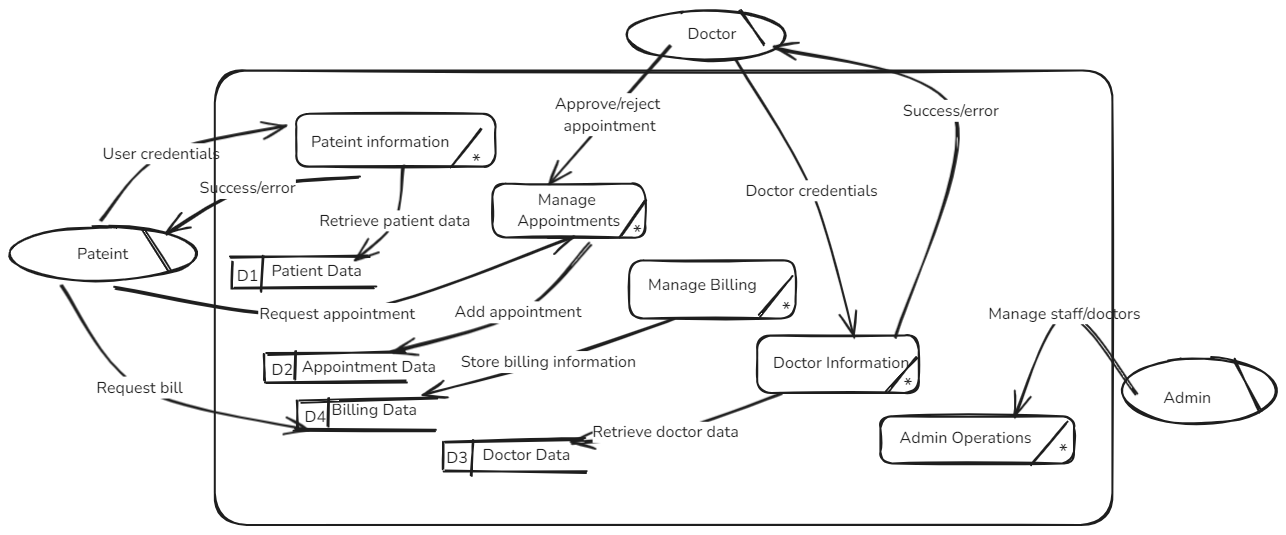
# **DESIGN**



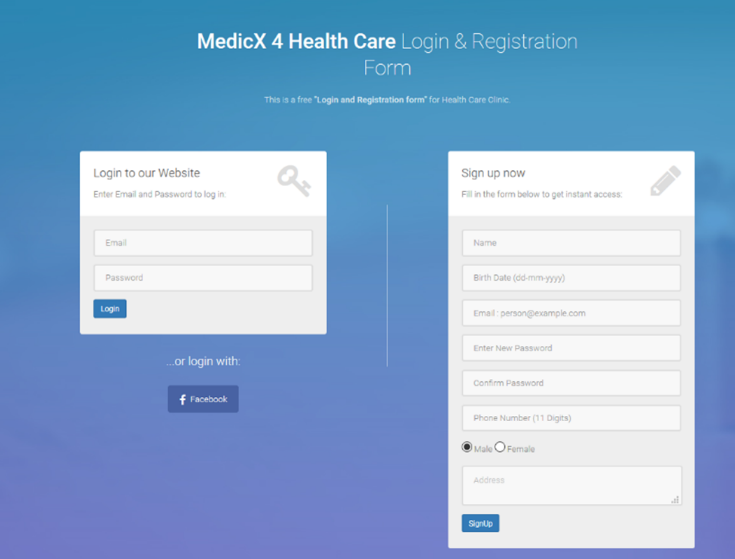
## **flow Diagram**



## **Data Flow Diagram**



## **User Interfaces**

**

*A screen shot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a medical survey

Description automatically generated*

*A screenshot of a medical application

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

**A screenshot of a medical registration form

Description automatically generated**

# **DESCRIPTION**

## **Technologies Use**

### **Frameworks and Platforms**

* ASP.NET for backend development.
* SQL Server for database management.

### **Programming Languages and Databases**

* C# for backend logic.
* SQL for database queries.

### **Frontend Technologies**

C#, HTML, Bootstrap, CSS, and JavaScript.

### **Backend Technologies**

ASP.NET framework for server-side logic.

### **Version Control and APIs**

* **GitHub Desktop**: For version control.

### **Operating Environment and Production Environment**

* **Development Environment**: Local machines with Windows.
* **Production Environment**: AWS servers for scalability and high availability.

## **Project Perspective/Context**

The CMS is developed to integrate and optimize clinic operations. It is designed to interface with third-party healthcare systems and electronic health records (EHRs) to provide a seamless experience.

## **Project Functionalities**

The CMS includes functionalities such as:

* **Patient Management:** Profile creation, appointment scheduling, and viewing treatment histories.
* **Doctor Management:** Managing schedules, patient records, and billing.
* **Administrative Oversight:** Viewing clinic statistics and managing user accounts.

## **User Classes and Characteristics**

## 

* **Patients:** Non-technical users requiring an intuitive interface.
* **Doctors:** Healthcare professionals with moderate technical expertise.
* **Administrators:** Technical users managing clinic operations.

# **REFACTORING METHODS**

namespace DBProject

{

public partial class AppointmentTaker : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

// Refactored to initialize the session and load available free slots in a separate method.

Session["freeSlot"] = "";

freeSlots(sender, e);

}

//---------------Function Called whenever a Free Slot is selected from the Grid View----//

protected void PAppointmentGrid\_RowCommand(object sender, GridViewCommandEventArgs e)

{

if (e.CommandName == "Select")

{

Int16 num = Convert.ToInt16(e.CommandArgument);

string appointment = PAppointmentGrid.Rows[num].Cells[2].Text;

string[] tokens = appointment.Split(':');

// Refactored to directly store free slot info in session

Session["freeSlot"] = tokens[0];

// Redirecting to appointment request confirmation

Response.BufferOutput = true;

Response.Redirect("AppointmentRequestSent.aspx");

return;

}

}

//-----------------------Function1--------------------------//

protected void freeSlots(object sender, EventArgs e)

{

// Refactored to encapsulate database access and handle free slot retrieval in a single method.

myDAL objmyDAl = new myDAL();

DataTable DT = new DataTable();

// Fetching Doctor and Patient ID from session variables

string dID1 = (string)Session["dID"];

int dID = Convert.ToInt32(dID1);

int pID = (int)Session["idoriginal"];

// Fetching free slots from DAL layer

int status = objmyDAl.getFreeSlots(dID, pID, ref DT);

// Refactored to handle different outcomes with a clearer messaging system

if (status == -1)

{

PAppointment.Text = "There was some error in retrieving the Doctors's Free Slots.";

}

else if (status == 0)

{

PAppointment.Text = "There is currently no free slot of this doctor.";

}

else if (status > 0)

{

PAppointment.Text = "The following are the " + status + " free slots of this doctor for today :";

PAppointmentGrid.DataSource = DT;

PAppointmentGrid.DataBind();

}

return;

}

//-----------------------Add a new function here------------------//

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using DBProject.DAL;

using System.Data;

namespace DBProject

{

public partial class TakeAppointment : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

// Refactored to initialize session and load department information in a single method.

Session["deptOriginal"] = "";

deptInfo(sender, e);

}

//---------------Function Called whenever a Department is selected from the Grid View----//

protected void TDeptGrid\_RowCommand(object sender, GridViewCommandEventArgs e)

{

if (e.CommandName == "Select")

{

Int16 num = Convert.ToInt16(e.CommandArgument);

string deptName = TDeptGrid.Rows[num].Cells[2].Text;

// Refactored to directly store department information in session

Session["deptOriginal"] = deptName;

// Redirecting to the view doctors page after department selection

Response.BufferOutput = true;

Response.Redirect("ViewDoctors.aspx");

return;

}

}

//-----------------------Function1--------------------------//

protected void deptInfo(object sender, EventArgs e)

{

// Refactored to encapsulate department information retrieval in a single method.

myDAL objmyDAl = new myDAL();

DataTable DT = new DataTable();

// Fetching department information using the data access layer (DAL)

int status = objmyDAl.getdeptInfo(ref DT);

// Refactored to provide specific messages based on the outcome of the department retrieval

if (status == -1)

{

TDept.Text = "There was some error in retrieving the Departments Information.";

}

else

{

TDept.Text = "Following are the departments available at our Clinic : ";

TDeptGrid.DataSource = DT;

TDeptGrid.DataBind();

}

return;

}}

# **TESTING**

### **Testing Technique**

Unit testing

Unit testing is a software testing technique where individual components or functions of a program are tested in isolation to ensure they work as expected. It helps identify bugs early, improves code reliability, and supports refactoring by providing a safety net of automated tests.

### **Test Cases**

**Class** AppointmentNotificationSent

**Function:** SendAppointmentRequest

### **Test cases Implementation with outputs**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using Moq;

using System.Web

using System.Web.SessionState;

namespace DBProject.Tests

{

[TestClass]

public class AppointmentNotificationSentTests

{

[DataTestMethod]

[DataRow(1, 1, 10, "Appointment request sent successfully")]

[DataRow(2, 1, 20, "There was some error in sending appointment request to the Doctor.")]

public void TestSendAppointmentRequest(int doctorID, int patientID, int slot, string expectedMessage)

{

// Arrange

var mockDAL = new Mock<myDAL>();

var objAppointmentNotification = new AppointmentNotificationSent();

// Mock the insertAppointment method of the DAL

mockDAL.Setup(m => m.insertAppointment(It.IsAny<int>(), It.IsAny<int>(), It.IsAny<int>(), ref It.Ref<string>.IsAny))

.Returns((int dID, int pID, int freeSlot, ref string message) =>

{

message = "Appointment request sent successfully"; // Simulating a success message

if (dID == 2) // Mocking failure for certain doctor ID

{

message = "There was some error in sending appointment request to the Doctor.";

return -1; // Simulate error

}

return 1; // Simulate success

});

// Act

string message = objAppointmentNotification.SendAppointmentRequest();

// Assert

Assert.AreEqual(expectedMessage, message);

}

}

}

6 CONCLUSIONS AND FURTHER WORK

The Clinic Management System significantly improves clinic operations and patient satisfaction. Future work includes:

* Integration with third-party healthcare systems.
* Mobile application development.
* Enhancements in analytics and reporting.

# **REFERENCES**

Projects’ GitHub Link

https://github.com/zuhaib1233/CMS

**THE END**