



KARACHI INSTITUTE OF ECONOMICS & TECHNOLOGY
College Of Engineering

Software Engineering

Complex Engineering Activity

Software Construction & Development

Student Name: _____

Student ID: _____

CEA Statement		Design and develop a C# project that addresses a real-world problem by implementing Object-Oriented Programming (OOP) principles. This project should be carefully planned, structured, and refactored to ensure maintainable, efficient, and clear code. The full Software Development Life Cycle (SDLC) will guide this process, and the final deliverable should include comprehensive documentation and a GitHub repository with a fully tested and functional project.					
PLO's		PLO1 – Engineering Knowledge		Bloom's Taxonomy	C2 – Understand		
		PLO3 – Design and Development			C3 – Apply		
		PLO9 – Team Work or Individual			A3 – Assume Responsibility		
		PLO11– Project Management			A5 – Organize		
COMPLEX ENGINEERING ACTIVITY							
CPA	CLO's	Aspects of Assessments	Excellent (75-100%)	Average (50-75%)	Poor (<50%)	Marks	
CEA-2 CEA-5	CLO1 PLO1	Understand: the associated concepts of Software Construction and Development from theory.	Complete understanding of the concepts / actively participate during lecture	Understand Software Development & Construction concepts / participate less in class	The student lacks a clear understanding of Software Development & Construction concepts completely and doesn't participate in class	Project Proposal (5%)	
						Problem Statement (5%)	
CEA-1 CEA-2 CEA-5	CLO5 PLO2	Designing and Constructing: Design and Construct optimized error-free, and reusable, code as a developer using the techniques of software construction, and also utilize software testing and debugging.	A complete solution / Explain necessary theories according to task description with great use of time and resource material.	The solution was complete but needed minor modifications/the student could have followed the specifications more closely.	The solution was complete but did not work, needed several modifications / did not make correct use of resource material or instructions.	Project Progress (40%)	
CEA-1	CLO3 PLO11	Report Management Report the outcome of an experiment/task in the standard format and Justify time and resource allocation to complete the assigned task	Accurately complete their Project Documentation and complete it on time.	Project Documentation completed but have some minor mistakes and time management also be effective	Project Documentation not completed but also not managing the time of the project.	Report (30%)	
CEA-2 CEA-5	CLO7 PLO9	Teamwork Completion of project tasks with proper teamwork and contribution.	Proactively work with other team members to complete assigned tasks.	Worked well with a team but did not offer much positive feedback.	Very little, if any, contributions to the group and less contribution in the completion of overall lab tasks.	Demo (20%)	
	Total Marks: 100						
	Scale Down: 30						

Remarks: _____

Complex Engineering Activity

The semester project is designed in a way to able students to solve a complex engineering Activity. The following characteristics of Complex Engineering Activity are targeted in this semester's project on Artificial Intelligence:

CEA-1	Range of resources	Diverse resources (people, money, equipment, materials, information, and technologies)
CEA-2	Level of interaction	Require resolution of significant problems arising from interactions between wide-ranging or conflicting technical, engineering, or other issues.
CEA-5	Familiarity	Can extend beyond previous experiences by applying principles-based approaches

Course	CPA Attributes	CLO	PLO (WA)	Bloom's Taxonomy
Software Construction & Development (Lab)	CEA-2 CEA-5	CLO1	PLO1	C2 (Understand)
	CEA-1 CEA-2 CEA-5	CLO5	PLO2	C3 (Apply)
	CEA-1	CLO3	PLO11	A5 (Organize)
	CEA-2 CEA-5	CLO7	PLO9	A3 (Assume Responsibility)

➤ CEA Statement:

Design and develop a C# project that addresses a real-world problem by implementing **Object-Oriented Programming (OOP)** principles. This project should be carefully planned, structured, and **refactored** to ensure maintainable, efficient, and clear code. The full Software Development Life Cycle (**SDLC**) will guide this process, and the final deliverable should include comprehensive documentation and a **GitHub repository** with a fully **tested** and functional project.

Project Requirements

1. Identify and Address a Real-World Problem:

- Choose a problem that can be addressed through software development. Ensure it is relevant, feasible, and impactful.

2. Use Object-Oriented Programming (OOP):

- Implement core OOP principles such as encapsulation, inheritance, polymorphism, and abstraction to structure the project logically and efficiently.

3. Project Planning and Design:

- Follow the SDLC methodology, moving through phases from requirements gathering, design, development, testing, deployment, and maintenance.
- Refactor your code during development to improve readability, performance, and maintainability.

4. Detailed Documentation:

- **Design Phase Documentation:** Create and document the following UML diagrams:
 - **Use Case Diagram:** Define actors and their interactions with the system.
 - **Class Diagram:** Detail the structure of your classes, relationships, and OOP principles.
 - **Data Flow Diagram (DFD):** Outline how data flows through the system.
 - **Sequence Diagram:** Describe interactions between objects or components in a process.

5. Development and Testing:

- After completing the design phase, proceed to the construction (coding) phase, ensuring that each module aligns with the design.
- Unit Testing: Validate each component's functionality with unit tests to confirm that the code is working as expected and handling edge cases appropriately.

6. Project Repository:

- Create a GitHub repository for the project.
- Organize your code, documentation, and diagrams within this repository.
- Include a README file with an overview, setup instructions, and usage guidelines for your project.

Submission Deliverables

- **Final Report:** A comprehensive report following the SDLC process, including:
 - Project overview and problem statement.
 - Detailed design phase diagrams (Use Case, Class, DFD, and Sequence Diagrams).
 - Code documentation and explanation of OOP principles used.
 - Testing report with unit test results.
- **GitHub Repository Link:** The complete project repository with structured folders for source code, documentation, diagrams, and tests.

➤ Project Phases

The project is carried out in three phases

1. Initial Study:

In the first phase, students are asked to bring the problems they intend to work on. Counseling is given to students in the lab and contact hours for finalizing their ideas and preparing a proposal. Students have to explore the problems/issues around them, which they can solve using programming constructs. If the problems brought by the students are irrelevant to the course or not feasible at this level, they are asked to bring up some other problem. Once ideas are finalized, constant counseling must provide by the Course Instructor/Lab Engr.

2. Project Proposal:

In this phase, students have to explore the literature or existing solutions for their selected project idea. In this phase, students are also encouraged to have a detailed analysis of the problem to solve it in a better way. Each student's project is unique and may have many possible solutions as well as may be explored and developed in a different way. After discussion, students are asked to submit a proposal on one idea approved by the Instructor/Lab Engr. A sample of the Project Proposal is also attached in the **Google Classroom**

3. Project Demonstration

Every project is checked by running and observing the output. Students had the option of using any programming tool of C# to develop a project. Students have to apply in-depth engineering knowledge to complete each project (**CEA1, CEA5**). During the initial study and formulation of the proposed solution, they focused on the detailed requirements (**CEA-2**), and real-time constraints (**CEA-2**) and performed an in-depth analysis.

A complete assessment of each student is presented in the result. Sample project reports are also attached in the **Google Classroom**

Projects were evaluated on the following criteria:

- **Problem Statement** (CEA-2, CEA-5) **5%**
- **Project Proposal** (CEA-2, CEA-5) **5%**
- **Project Progress** (CEA-1, CEA-2, CEA-5) **40%**
- **Report** (CEA-1) **30%**
- **Demo** (CEA-2, CEA-5) **20%**

➤ Summary:

The following are salient outcomes of the semester project in terms of Complex Engineering Activity:

- Brainstorming exercises forced them to explore the surrounding environment to sort out the problems to be solved using programming constraints.
- Problem formulation enhances their ability to gather real-time requirements and address conflicts/constraints.
- Design/Implementation gave them a chance to go through the in-depth engineering knowledge to solve the problem and analyze it in an effective way

Final Project Report

Course: Software Construction & Development

Hospital Management System

Submitted By

Muhammad Hamza (Group Leader)

FA21-14394

Asad Rasheed

FA21-14230

Department of Software Engineering



KARACHI INSTITUTE OF ECONOMICS & TECHNOLOGY
College of Engineering

ABSTRACT

The hospital management system provides a solution to streamline and automate the operations and key administrative in a healthcare department. This project aims to provide that level of functionality with a user-friendly interface for managing hospital operations such as patient management, room management, staff management, checkouts and more.

This project is made by utilizing the concepts of OOP in the C# language with the backend support of SQL server for data management. This system allows the hospital staff to manage patients, staff, rooms and medical records. Some of the system's functionalities are patient registration, information retrieval, room assignments, staff management and checkout processes. The windows form-based GUI with features like data fetching and efficient data storage improves the overall system.

The system allows managing hospitals' everyday business without infringing on security and data integrity. Altogether, the project proposed in the system gives a good example of being an easily extensible and modified management solution that suits each given medical facility with unique requests.

Table Of Contents

ABSTRACT	2
CHAPTER # 1 INTRODUCTION TO PROBLEM.....	4
1.1 Introduction	5
Problem Statement.....	5
1.2 Purpose.....	5
Project Significance	6
1.3 Objective	6
1.4 Existing solution	6
1.5 Purposed Solution	6
1.6 Scope.....	7
CHAPTER # 2 LITERATURE REVIEW	8
2.1 Literature Review	9
CHAPTER # 3 PROBLEM DEFINITION & REQUIREMENT ANALYSIS	11
3.1 Problem Definition.....	12
3.2 Requirement Analysis	12
CHAPTER # 4 DESIGN & IMPLEMENTATION	14
4.1 Data Flow Chart.....	15
4.2 Implementation	21
CHAPTER # 5 TESTING	42
5.1 Testing Procedure & Test Cases:.....	43
5.2 Results	50
CHAPTER # 6 FUTURE ENHANCEMENTS.....	51
6.1 Future Enhancements	52
CHAPTER # 7 APPENDIX	53

CHAPTER # 1 INTRODUCTION TO PROBLEM

1.1 Introduction

The hospital management system aims to simplify the daily tasks of hospital management like room availability, processes of billing, patient records and staff details. The system provides a user-friendly application for staff and admins to manage most of the hospital tasks in one place and removing any manual effort. By introducing this system, the hospitals can improve efficiency, reduce human errors and ensure better care for the patients.

Problem Statement

Nowadays, hospitals struggle with handling bulk data of patients, rooms and staff using outdated and manual processes. This leads to overall inefficiency in most of the operations like delays, data loss and errors. For example, searching records for a patient or room might take long time or misplaced paper records of a patient could cause delays in patient care. These factors not only increase the work load but also reduce the quality of healthcare services.

1.2 Purpose

The purpose of this document is to highlight the requirements of hospital management system. This is a windows form application made in C# and SQL. We want to create a centralized, user-friendly and efficient platform for managing hospital operations. This document will describe the functionality of the system in detail and highlight the dependencies that exist if any.

Project Significance

The project has been developed using the standard software development methodology and standard processes of project management, details can be found in the next section.

The developer will make use of mature technology frameworks; the following underlying technologies used, SQL Server Management Studio and C#.

1.3 Objective

The objective of the project is to provide a system which can help the hospitals to automate the processes such as patient management which includes registering, updating and managing checkouts for patients. Staff management by storing and retrieving staff details like names, roles and contact information. Room management by allocating and managing the available rooms. Other necessary things include such as data accuracy and system efficiency.

1.4 Existing solution

Most of the hospitals have recorded information through manual, paper-based record systems or at best basic spreadsheets pertaining to patients, staffs, and rooms. Several large healthcare facilities utilize leading-edge hospital management software while such solutions are mostly out of the reach of cost-prohibitive, complex and extremely extensive training requirements for such services. Most smaller hospitals and clinics cannot afford such high technology and continue with inefficient and fallible manual processes, more error-prone, vulnerable to misplaced data, and ultimately delayed patient care.

1.5 Purposed Solution

The proposed custom hospital management system is affordable, easy to use and handles the small healthcare needs in a hospital. This system will automate many key

processes of the hospitals. It is built using C# windows forms and SQL Server. With minimal training of staff, it can be utilized effectively. It provides ease of use and flexibility.

1.6 Scope

The application provides quick patient registrations, updating their details and tracking their history. The staff details, roles and salaries can be stored effectively. Tracking of room availability and assignments are provided with their status. Patient checkouts are being handled with updating the room status and final bills. Data security and user roles have been prioritized to handle the sensitive information of any individual.

CHAPTER # 2 LITERATURE REVIEW

2.1 Literature Review

Paper # 1:

Hsieh, S. L., Lai, F., Cheng, P. H., Chen, J. L., Lee, H. H., Tsai, W. N., ... Chen, C. H. (2006). *An Integrated Healthcare Enterprise Information Portal and Healthcare Information System Framework. 2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. doi:10.1109/iembs.2006.260715

This paper includes a healthcare enterprise information portal with a hospital information system which addresses system issues and reduce operational costs. Some advancements used are middle ware-based integrations, CRM and other supports like e-learning and vaccination tracking.

Paper # 2:

Qingzhang Chen, Jie Chen, Li, Y., & Fei Xu. (2010). *Design and implement of performance management system for hospital staff based on BSC. 2010 International Conference on Networking and Digital Society*. doi:10.1109/icnds.2010.5479261

This paper proposed a hospital performance management system for the staff using the balanced scorecard approach. It integrates data and enables the automated KPI collection and quantitative evaluation for staff performance. The system enhances and ensures alignment with hospital strategic goals.

Paper # 3:

Yang, T. H., Cheng, P. H., Yang, C. H., Lai, F., Chen, C. L., Lee, H. H., ... Sun, Y. S. (2006). *A Scalable Multi-tier Architecture for the National Taiwan University Hospital Information System based on HL7 Standard. 19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*. doi:10.1109/cbms.2006.27

This paper talks about a multi-tier scalable architecture for the hospital management system. It shows the use of 4 layers switches for load balancing, data consistency and high availability which ensures stable performance over heavy loads of data. This approach can be used for larger hospital management systems.

Paper # 4:

Wangbin, Xieqi, Shihuaxin, Caoxinyu, Wangwenjing, & Chendi. (2014). *The design and implementation of inpatient medical expenses analysis system. 2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. doi:10.1109/bibm.2014.6999350

This paper shows the development of medical expense analysis system which provides the statistical data for hospital admins. This allows the administrations to gain insights on the hospital operations and then provide the effective feedback accordingly.

Paper # 5:

Perdanakusuma, D., Puspitasari, W., & Saputra, M. (2020). *Utilizing Open ERP for Creating Medical Record Management System in Smart Hospital : A Case Study. 2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. doi:10.1109/iaict50021.2020.91720

This paper shows the development of medical record management system using open ERP for smart hospitals. The unified processes of ERP system and the centralized DB enhances efficiency and data accessibility. It highlights the importance and potential of ERP systems in the field of healthcare.

CHAPTER # 3 PROBLEM DEFINITION & REQUIREMENT ANALYSIS

3.1 Problem Definition

Managing hospitals manually can be a very frustrating task. The existing manual processes like managing patient records, room allocations, staff details and patient checkouts are all inefficient, can have human errors and time consuming. This will result in delayed patient care, mismanagement of rooms and unnecessary workload on hospital staff. The hospital management system offers and tackles all these issues by providing a one platform solution.

The vision for this system is to provide centralization, and user-friendly automation of key hospital operations, such as patient registration, room allocations, staff management, and checkouts. Hospitals will be able to eliminate human errors, increase their operational efficiency, and improve patient care through this system. The desktops and laptops connected with a local or online server will provide 24/7 access to this system. With the ensured safe storage of data and smooth access to information, hospital administration will save time while reducing costs. This software system is going to be a one-stop center from which all users, including hospital administrators, nurses, and the staff, will be accessing, managing, and updating all the relevant data available in the hospitals. Its interface will be user-friendly and attractive in order that all the stakeholders have a smooth usage experience. The system would provide access functionalities for relevant information for both staff and patients while maintaining security and privacy.

3.2 Requirement Analysis

Following use cases shows the requirements of this hospital management system. The main actors of this system are hospital admins, staff and patients:

1. Patient Management

The admins of the system can register new patients, update their details and check their medical records. This involves adding a new patient, assigning a room and recording the patient details like name, medical conditions, age etc.

2. Staff Management

The admins of the system can store and retrieve staff information which includes their names, gender, contact details, salary and job roles.

3. Room Management

The admins of the system can manage the status and availability of hospital rooms like occupied, vacant or under maintenance. The process is handled as the part of patient registration process.

4. Patient Checkout Management

The admins of the system will handle the discharged patient by updating their status, releasing the assigned room and finalizing the bills.

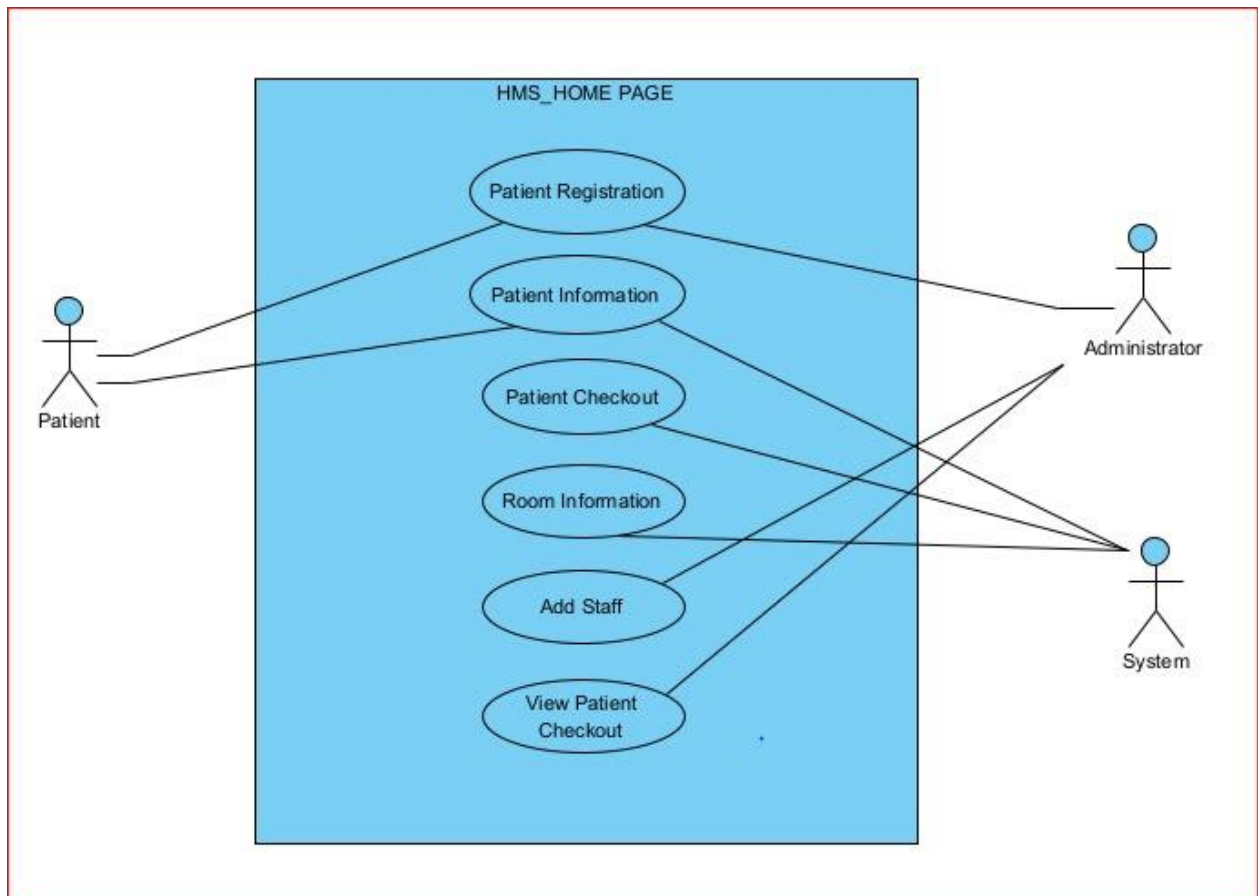
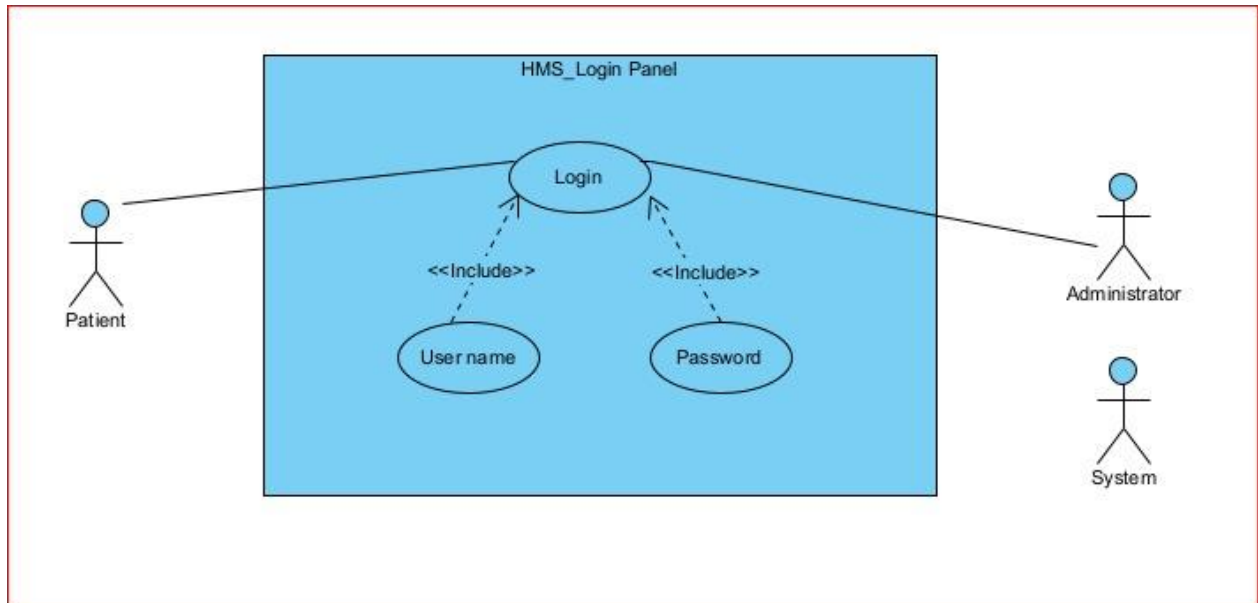
5. Other Requirements

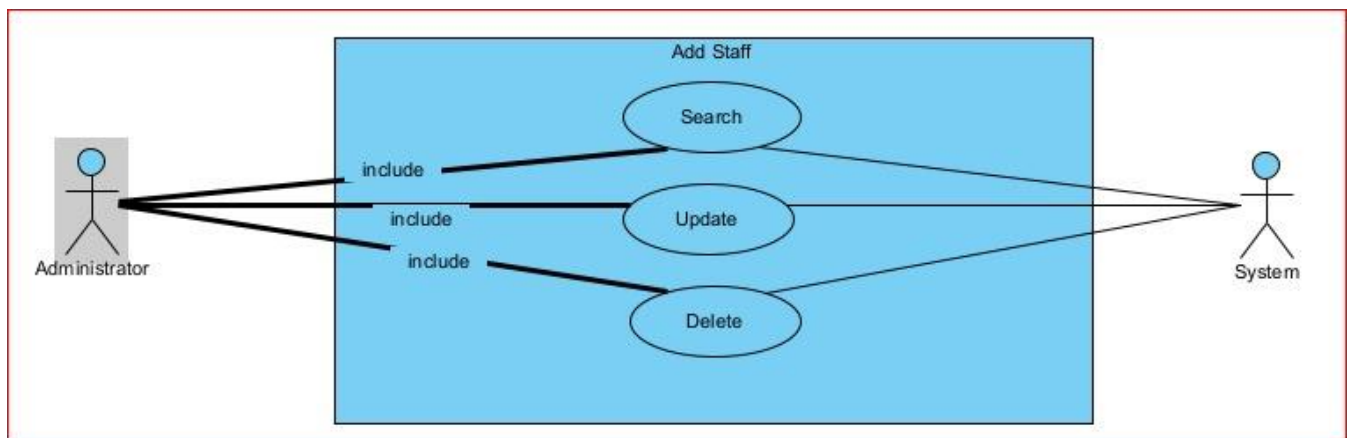
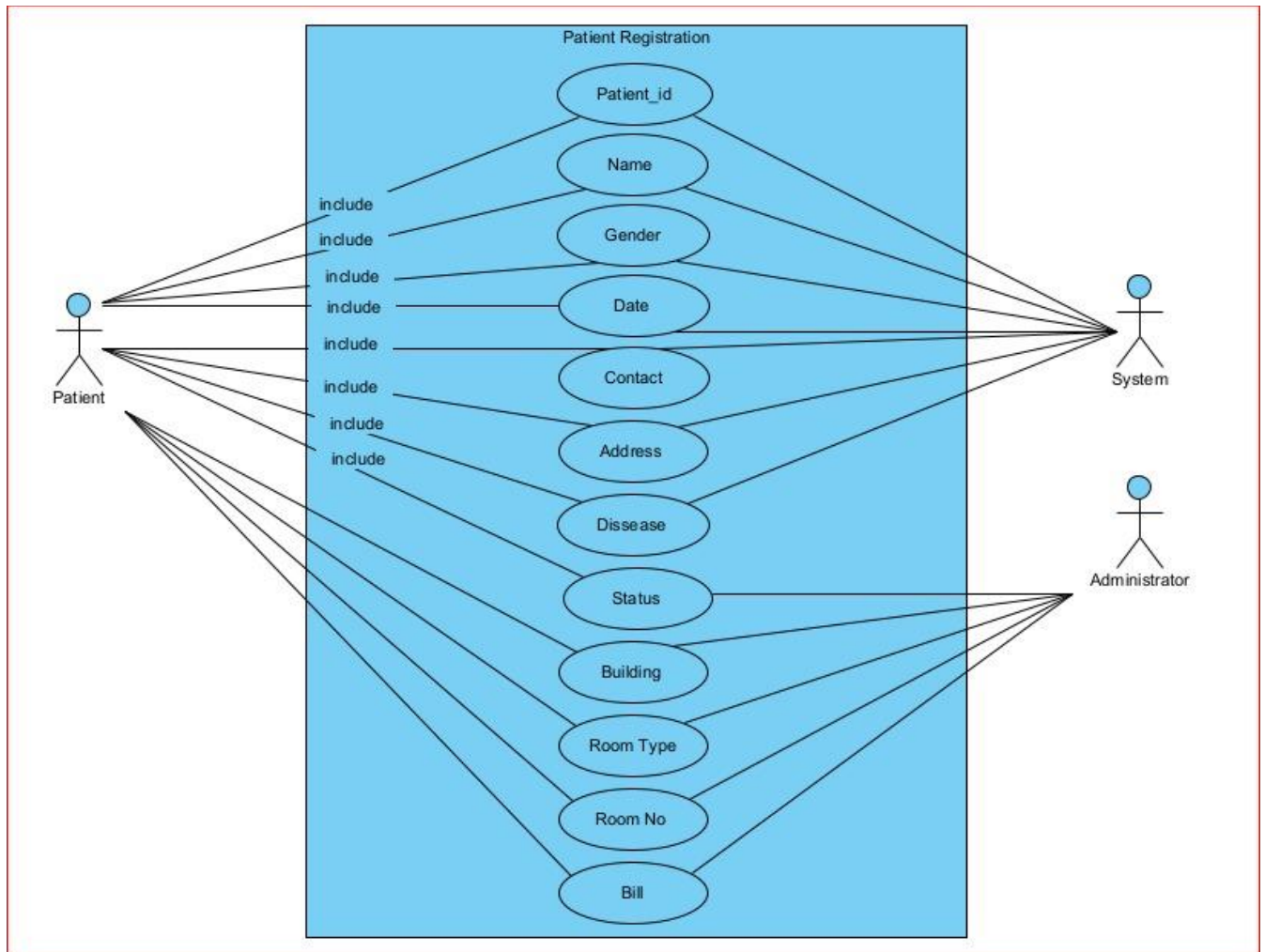
These includes the non-functional requirements such as usability, reliability, scalability and security of the system. These will ensure the other processes to keep running so the management system can run effectively.

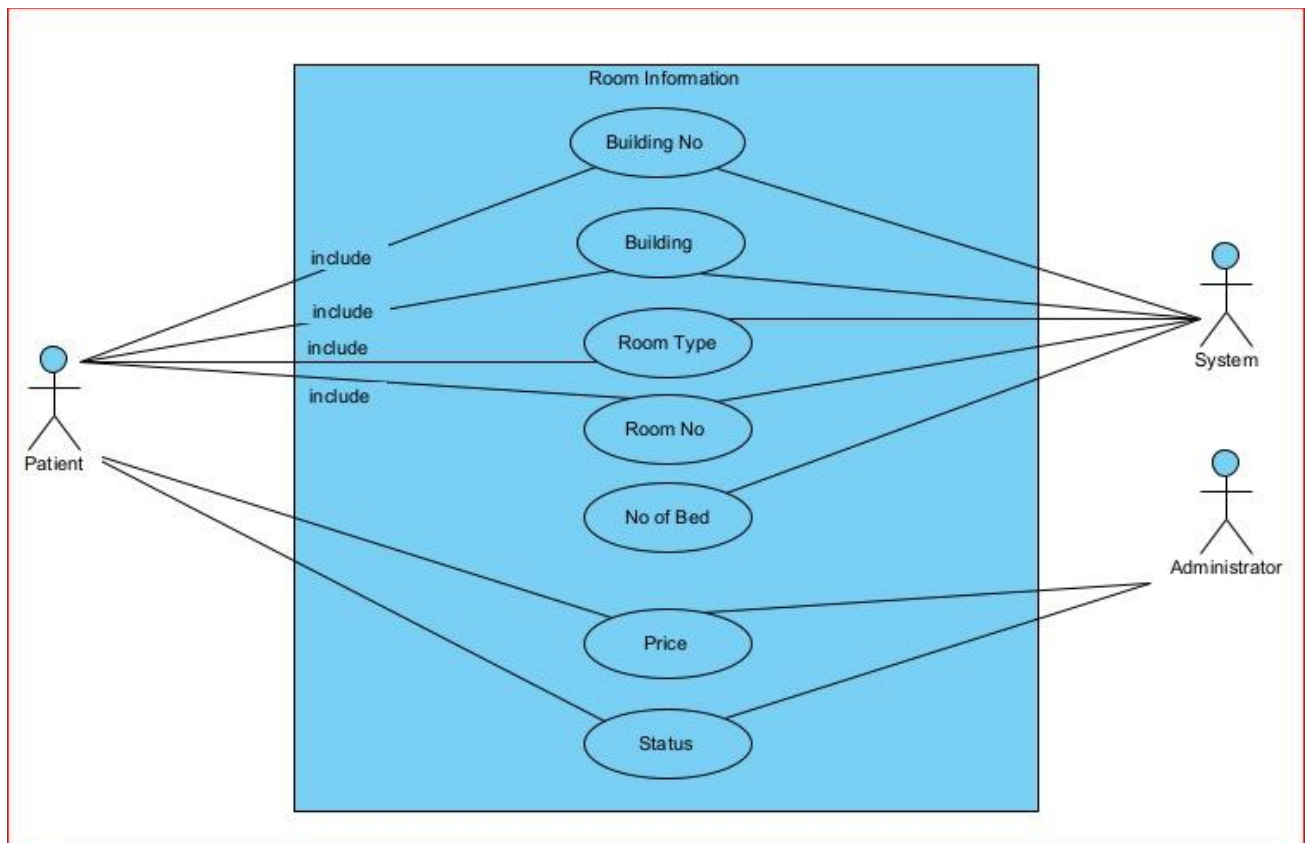
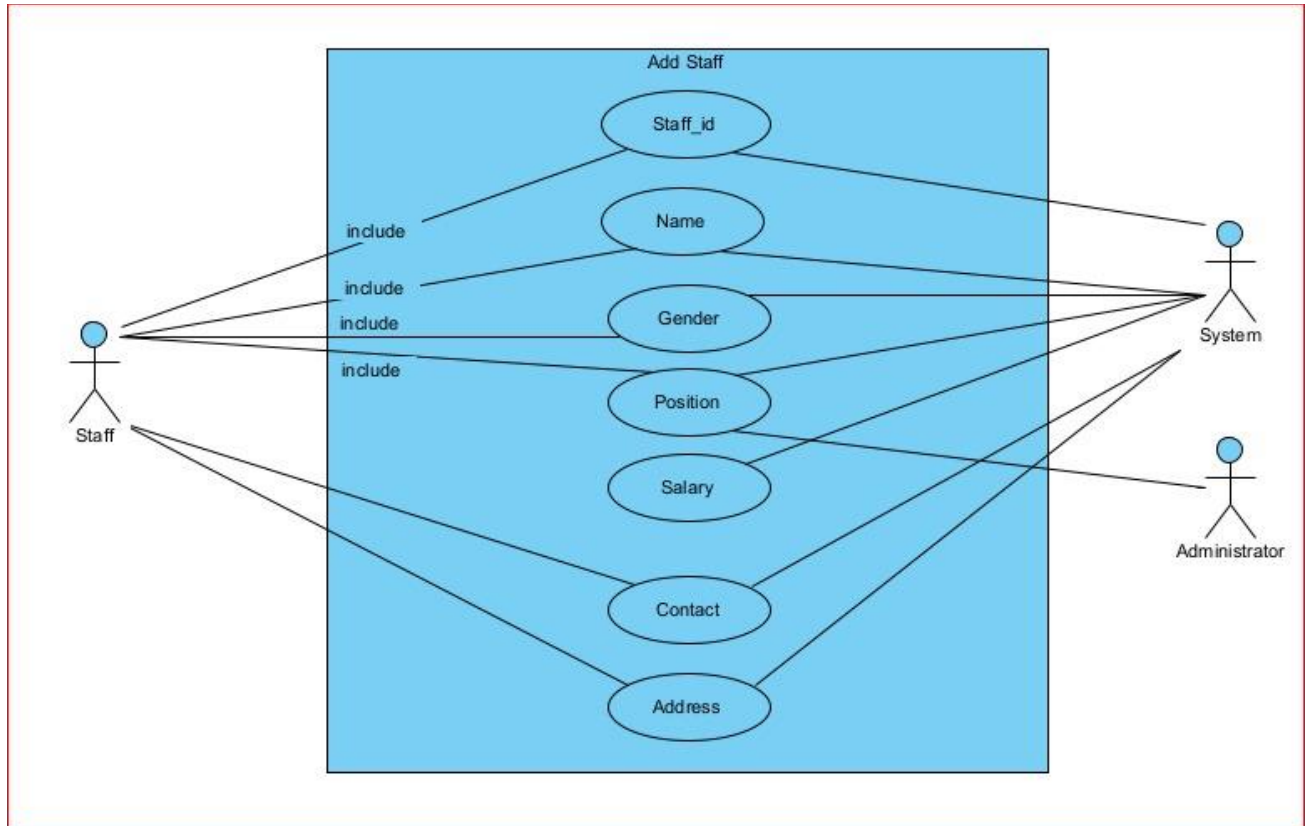
CHAPTER # 4 DESIGN & IMPLEMENTATION

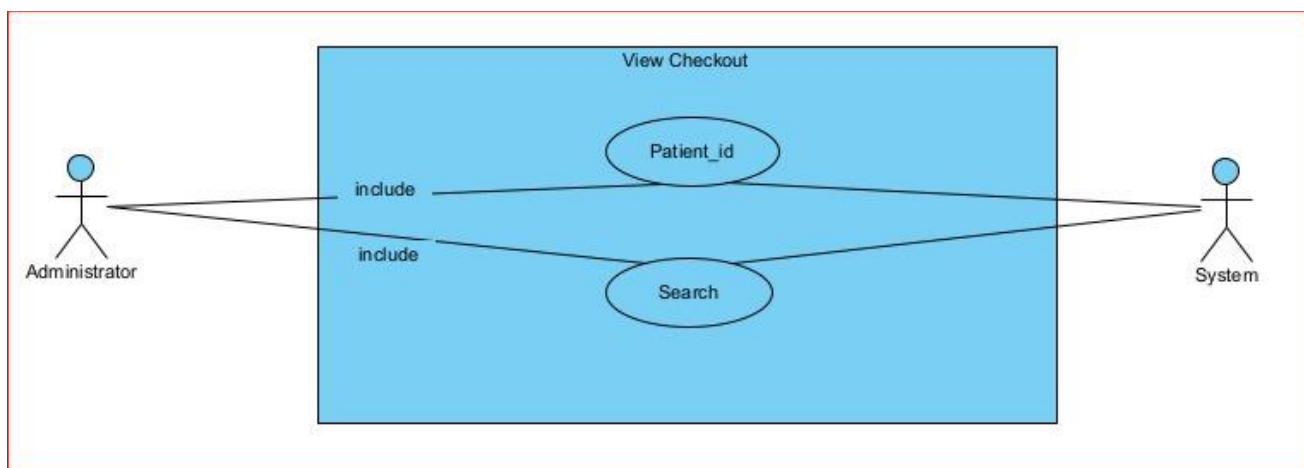
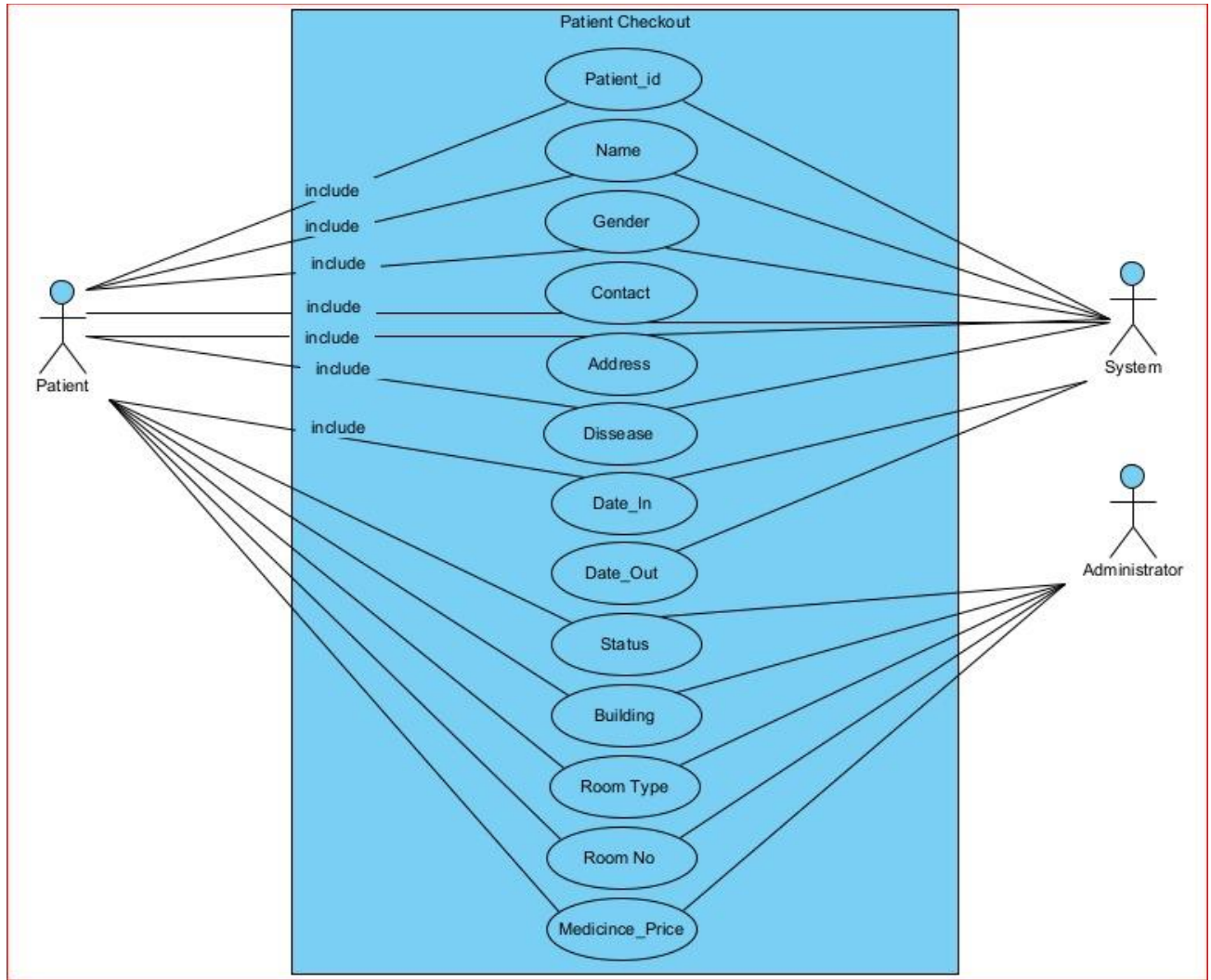
4.1 Data Flow Chart

Use Case Diagram:

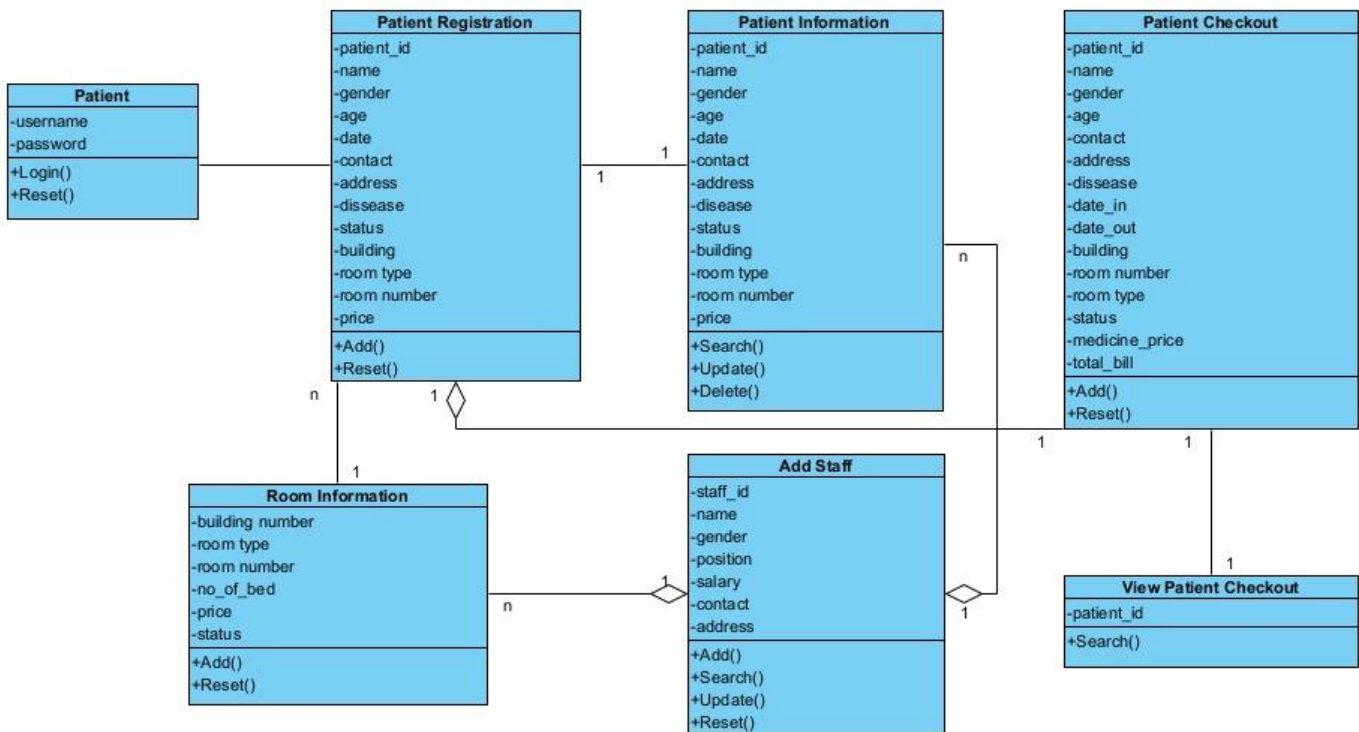




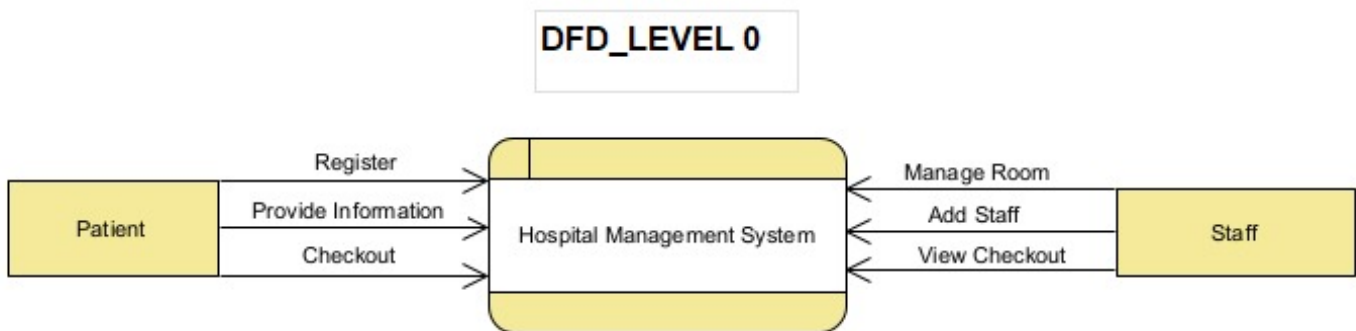


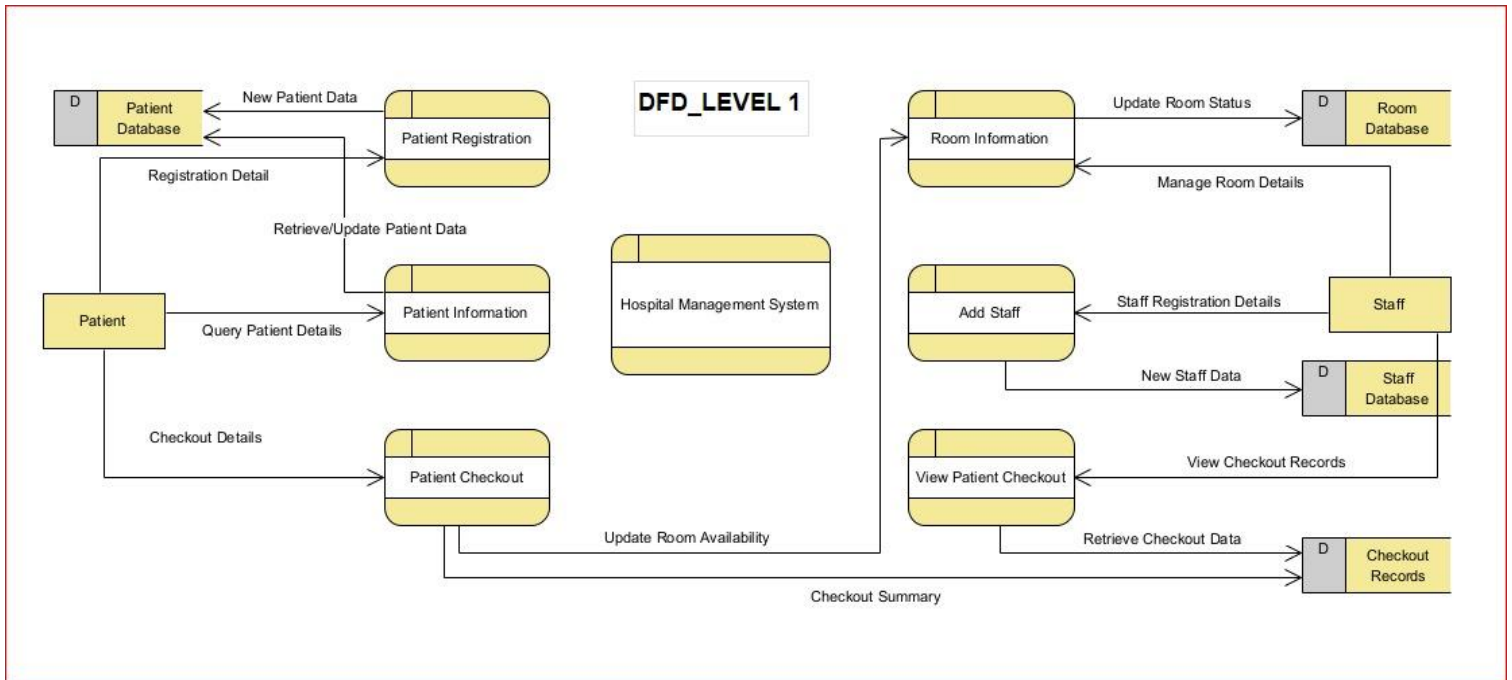


Class Diagram:

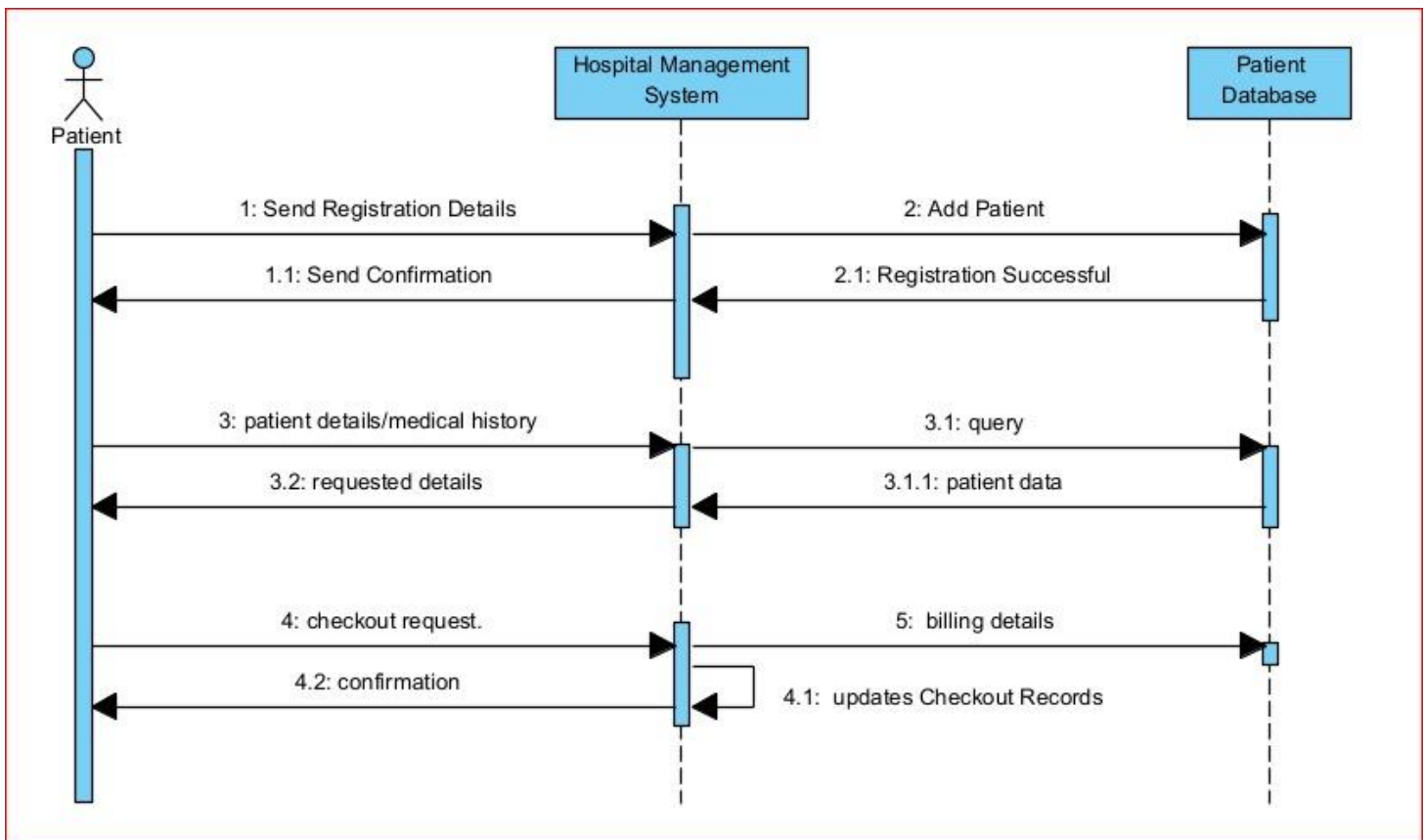


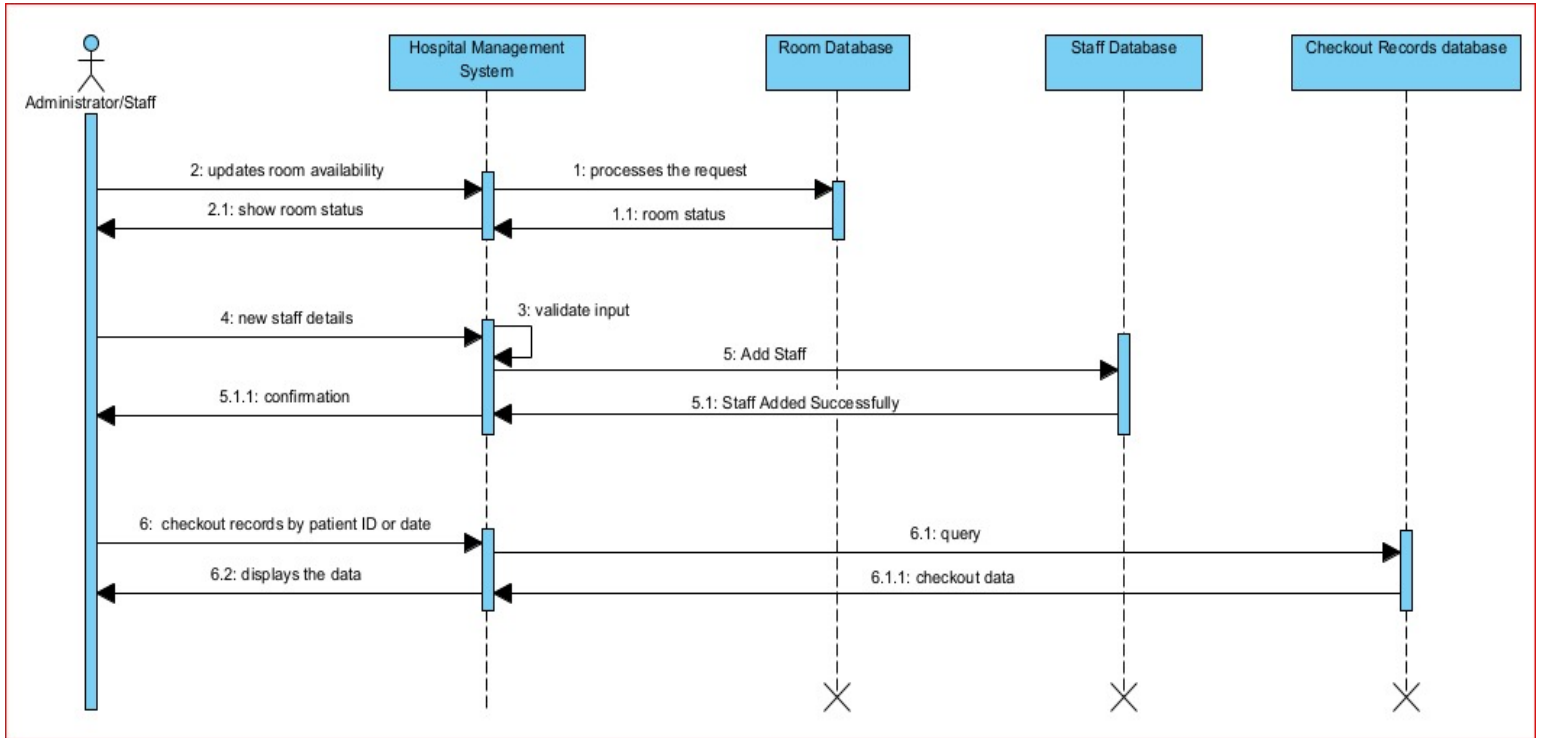
DFD Diagram:





Sequence Diagram:





4.2 Implementation

The main codes of the windows forms of different operations are given below:

Login Form

```

using HospitalManagementSystemCSharp.Refactoring;
using System;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class Login : Form
    {
        //Call the interface
        private readonly ILoginBehavior _loginBehavior;

        public Login()
        {
            InitializeComponent();
            _loginBehavior = new LoginBehavior(this);
        }

        //Handle login method
        private void Button1_Click(object sender, EventArgs e)
        {

```

```

        _loginBehavior.HandleLogin();
    }

    //Handle clear fields method
    private void Button2_Click(object sender, EventArgs e)
    {
        _loginBehavior.ClearInputFields();
    }

    //Encapsulated fields for textBox1 and textBox2
    public string Username => textBox1.Text;
    public string Password => textBox2.Text;

    //Method to clear text fields
    public void ClearInputFields()
    {
        textBox1.Text = "";
        textBox2.Text = "";
    }

    //Method to show a success message
    public void ShowSuccessMessage(string message)
    {
        MessageBox.Show(message);
    }

    //Method to show an error message
    public void ShowErrorMessage(string message)
    {
        MessageBox.Show(message);
    }

    //Method to navigate to Home form
    public void OpenHomePage()
    {
        this.Visible = false;
        Home homePage = new Home();
        homePage.ShowDialog();
    }
}
}

```

The screenshot shows a window titled "Login-HMS". Inside the window, there is a light blue background. At the top center, there is a circular icon containing a green grid with a white cross in the center. Below the icon, the text "Login Panel" is displayed in orange. Underneath, there are two input fields: "Username" with the text "admin" and "Password" with four asterisks "****". At the bottom, there are two buttons: "Login" and "Clear".

Home (Dashboard) Form

```
using HospitalManagementSystemCSharp.Refactoring;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class Home : Form
    {
        //Getting the interfaces for each page to handle their functionality
        private readonly IPatientService _patientService;
        private readonly IRoomInfoRepository _roomInfoRepository;
        private readonly IStaffRepository _staffRepository;
        private readonly ICheckoutRepository _checkoutRepository;

        //Initializng the dashboard
        public Home()
        {
            InitializeComponent();
            _patientService = new PatientService();
            _roomInfoRepository = new RoomInfoRepository();
            _staffRepository = new StaffRepository();
            _checkoutRepository = new CheckoutRepository();
        }
    }
}
```

```

e) //Goto the patient registration page
private void PatientRegistrationToolStripMenuItem_Click(object sender, EventArgs
{
    PatientRegistration obj = new PatientRegistration(_patientService);
    obj.ShowDialog();
}

//Goto the patient info page
private void PatientInformationToolStripMenuItem_Click(object sender, EventArgs e)
{
    PatientInformation obj1 = new PatientInformation();
    obj1.ShowDialog();
}

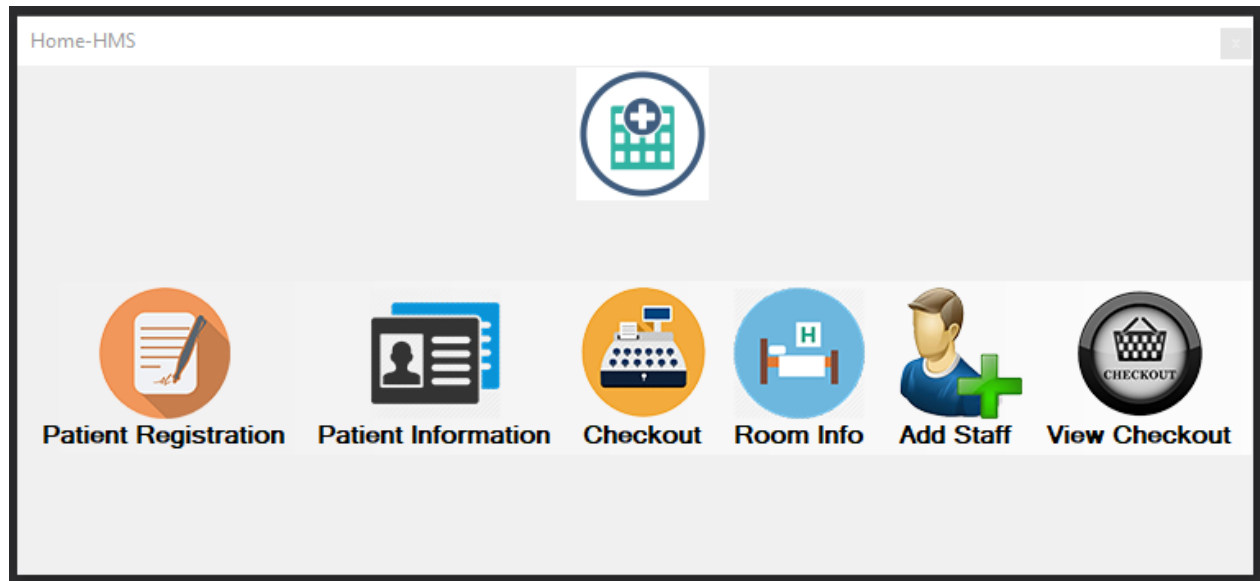
//Goto the patient checkout page
private void CheckoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    PatientCheckOut obj2 = new PatientCheckOut();
    obj2.ShowDialog();
}

//Goto the room info page
private void RoomInfoToolStripMenuItem_Click(object sender, EventArgs e)
{
    RoomInfo obj3 = new RoomInfo(_roomInfoRepository);
    obj3.ShowDialog();
}

//Goto the staff info page
private void AddStaffToolStripMenuItem_Click(object sender, EventArgs e)
{
    StaffInformation obj4 = new StaffInformation(_staffRepository);
    obj4.ShowDialog();
}

//Goto the view checkout page
private void ViewCheckoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    ViewPatientCheckOut obj5 = new ViewPatientCheckOut(_checkoutRepository);
    obj5.ShowDialog();
}
}
}

```



Patient Checkout Form

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using HospitalManagementSystemCSharp.Refactoring;

namespace HospitalManagementSystemCSharp
{
    public partial class PatientCheckOut : Form, IPatientCheckOut
    {
        private readonly PatientCheckOutRepository _repository = new
        PatientCheckOutRepository();

        public PatientCheckOut()
        {
            InitializeComponent();

            //Method to load patient details
            public void LoadPatientDetails(int patientId)
            {
                _repository.LoadPatientDetails(patientId, out string name, out string gen, out
                string age, out string contact, out string addr, out string disease);

                if (!string.IsNullOrEmpty(name))
                {
                    textBox2.Text = name;
                    radioButton1.Checked = gen == "Male";
                }
            }
        }
    }
}
```

```

        radioButton2.Checked = gen == "Female";
        textBox3.Text = age;
        textBox5.Text = contact;
        textBox6.Text = addr;
        textBox7.Text = disease;
    }
    else
    {
        MessageBox.Show($"Sorry, patient with ID {patientId} is not available.");
        textBox1.Clear();
    }
}

//Method to save patient checkout information
public void SavePatientCheckout()
{
    string gender = radioButton1.Checked ? "Male" : "Female";
    _repository.SavePatientCheckout(textBox2.Text, gender, textBox3.Text,
textBox5.Text, textBox6.Text, textBox7.Text, textBox8.Text, textBox9.Text, textBox10.Text,
textBox11.Text, textBox12.Text, textBox14.Text, textBox13.Text, textBox15.Text);

    MessageBox.Show("Patient Checkout Information Saved Successfully.");
    ClearForm();
}

//Method to clear the form fields
public void ClearForm()
{
    textBox2.Clear();
    textBox3.Clear();
    textBox5.Clear();
    textBox6.Clear();
    textBox7.Clear();
    textBox8.Clear();
    textBox9.Clear();
    textBox10.Clear();
    textBox11.Clear();
    textBox12.Clear();
    textBox13.Clear();
    textBox14.Clear();
    textBox15.Clear();
}

//Event handler for patient ID text box changes
private void TextBox1_TextChanged(object sender, EventArgs e)
{
    if (int.TryParse(textBox1.Text, out int patientId))
    {
        LoadPatientDetails(patientId);
    }
    else
    {
        MessageBox.Show("Please enter a valid patient ID.");
    }
}

//Event handler for save button click

```

```

private void Button1_Click(object sender, EventArgs e)
{
    SavePatientCheckout();
}

//Event handler for clear button click
private void Button2_Click(object sender, EventArgs e)
{
    ClearForm();
}
}
}

```

Patient Information Form

```

using HospitalManagementSystemCSharp.Refactoring;
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class PatientInformation : Form, IPatientRepository

```

```

{
    private readonly string _connectionString = "Data
Source=localhost\\SQLEXPRESS;Initial Catalog=hms;Integrated Security=True";

    public PatientInformation()
    {
        InitializeComponent();
    }

    //Method to load patient data into the DataGridView
    public void LoadPatientData()
    {
        string query = "SELECT * FROM patient";
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            SqlCommand cmd = new SqlCommand(query, con);
            SqlDataAdapter da = new SqlDataAdapter(cmd);
            DataTable dt = new DataTable();
            da.Fill(dt);
            dataGridView1.DataSource = new BindingSource(dt, null);
        }
    }

    //Method to search for a patient by ID and display details
    public void SearchPatientById(int patientId)
    {
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            string query = "SELECT * FROM patient WHERE id = @id";
            SqlCommand cmd = new SqlCommand(query, con);
            cmd.Parameters.AddWithValue("@id", patientId);

            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.Read())
            {
                textBox2.Text = reader["name"].ToString();
                radioButton1.Checked = reader["gen"].ToString() == "Male";
                radioButton2.Checked = reader["gen"].ToString() == "Female";
                textBox3.Text = reader["age"].ToString();
                textBox4.Text = reader["date"].ToString();
                textBox5.Text = reader["cont"].ToString();
                textBox6.Text = reader["addr"].ToString();
                textBox7.Text = reader["disease"].ToString();
                textBox8.Text = reader["status"].ToString();
                textBox10.Text = reader["r_type"].ToString();
                textBox9.Text = reader["building"].ToString();
                textBox11.Text = reader["r_no"].ToString();
                textBox12.Text = reader["price"].ToString();
            }
            else
            {
                MessageBox.Show($"Sorry, patient with ID {patientId} is not
available.");
            }
            con.Close();
        }
    }
}

```



```

    }

    //Method to update patient details in the database
    public void UpdatePatient()
    {
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            string gender = radioButton1.Checked ? "Male" : "Female";
            string updateQuery = "UPDATE patient SET name=@name, gen=@gen, age=@age,
date=@date, cont=@cont, addr=@addr, disease=@disease, status=@status, r_type=@rType,
building=@building, r_no=@rNo, price=@price WHERE id=@id";

            SqlCommand cmd = new SqlCommand(updateQuery, con);
            cmd.Parameters.AddWithValue("@name", textBox2.Text);
            cmd.Parameters.AddWithValue("@gen", gender);
            cmd.Parameters.AddWithValue("@age", textBox3.Text);
            cmd.Parameters.AddWithValue("@date", textBox4.Text);
            cmd.Parameters.AddWithValue("@cont", textBox5.Text);
            cmd.Parameters.AddWithValue("@addr", textBox6.Text);
            cmd.Parameters.AddWithValue("@disease", textBox7.Text);
            cmd.Parameters.AddWithValue("@status", textBox8.Text);
            cmd.Parameters.AddWithValue("@rType", textBox10.Text);
            cmd.Parameters.AddWithValue("@building", textBox9.Text);
            cmd.Parameters.AddWithValue("@rNo", textBox11.Text);
            cmd.Parameters.AddWithValue("@price", textBox12.Text);
            cmd.Parameters.AddWithValue("@id", textBox1.Text);

            cmd.ExecuteNonQuery();
            MessageBox.Show($"Patient {textBox2.Text}'s details updated
successfully.");
            con.Close();
            // Refresh the patient list after update
            LoadPatientData();
        }
    }

    //Method to delete a patient record
    public void DeletePatient(int patientId)
    {
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            string deleteQuery = "DELETE FROM patient WHERE id = @id";
            SqlCommand cmd = new SqlCommand(deleteQuery, con);
            cmd.Parameters.AddWithValue("@id", patientId);
            cmd.ExecuteNonQuery();
            con.Close();
            MessageBox.Show("Patient record deleted successfully.");
            // Refresh the patient list after deletion
            LoadPatientData();
        }
    }

    //Event handler for form load
    private void PatientInformation_Load(object sender, EventArgs e)
    {

```

```

        // Load patient data on form load
        LoadPatientData();
    }

    //Event handler for search button click
    private void Button1_Click(object sender, EventArgs e)
    {
        if (int.TryParse(textBox1.Text, out int patientId))
        {
            SearchPatientById(patientId);
        }
        else
        {
            MessageBox.Show("Please enter a valid patient ID.");
        }
    }


    //Event handler for update button click
    private void Button2_Click(object sender, EventArgs e)
    {
        UpdatePatient();
    }

    //Event handler for delete button click
    private void Button3_Click(object sender, EventArgs e)
    {
        if (int.TryParse(textBox1.Text, out int patientId))
        {
            DeletePatient(patientId);
        }
        else
        {
            MessageBox.Show("Please enter a valid patient ID.");
        }
    }
}
}
}

```

PatientInformation

Patient Information



Patient Id:
 Disease:

Name:
 Status:

Gender: ☒ Male ☐ Female
 Building:

Age:
 Room Type:

Date:
 Room No:

Contact:
 Price:

Address:

	Id	name	gen	age	date	cont
▶	15	Hamza	Male	21	12/12/12	12345678
*						

Patient Registration Form

```

using System;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class PatientRegistration : Form
    {
        private readonly IPatientService _patientService;
    }
}

```

```

//Constructor to initialize the form
public PatientRegistration(IPatientService patientService)
{
    _patientService = patientService;
    InitializeComponent();
}

//Event handler for the 'Save' button click
private void Button1_Click(object sender, EventArgs e)
{
    try
    {
        string gender = GetGender();
        _patientService.SavePatientInformation(gender, textBox2.Text,
textBox3.Text, textBox4.Text,
                                                    textBox5.Text, textBox6.Text,
textBox7.Text, textBox8.Text,
                                                    textBox10.Text, textBox9.Text,
textBox11.Text, textBox12.Text);
        MessageBox.Show("Patient Information Saved Successfully..");
        ClearFormFields();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Event handler to load the patient registration form
private void PatientRegistration_Load(object sender, EventArgs e)
{
    try
    {
        textBox1.Text = _patientService.GetNextPatientId();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Event handler for the 'Clear' button click
private void Button2_Click(object sender, EventArgs e)
{
    ClearFormFields();
}

//Method to get the gender based on radio button selection
private string GetGender()
{
    return radioButton1.Checked ? "Male" : "Female";
}

//Method to clear all form fields
private void ClearFormFields()
{
    textBox1.Clear();
}

```

```

        textBox2.Clear();
        textBox3.Clear();
        textBox4.Clear();
        textBox5.Clear();
        textBox6.Clear();
        textBox7.Clear();
        textBox8.Clear();
        textBox9.Clear();
        textBox10.Clear();
        textBox11.Clear();
        textBox12.Clear();
    }
}

```

The screenshot shows a Windows application window titled "PatientRegistration". Inside the window, there is a light blue background with the title "Register New Patient" in red text at the top center. To the right of the title is a circular icon containing a green grid with a white plus sign. The form consists of two columns of labels and text boxes. The left column includes labels for Patient Id, Name, Gender, Age, Date, Contact, and Address. The right column includes labels for Disease, Status, Building, Room Type, Room No, and Price. The Patient Id text box contains the number "16". The Gender label has two radio buttons, with "Male" selected. At the bottom of the form, there are two buttons labeled "Add" and "Reset".

Room Info Form

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using HospitalManagementSystemCSharp.Refactoring;

namespace HospitalManagementSystemCSharp
{
    public partial class RoomInfo : Form
    {
        private readonly IRoomInfoRepository _repository;

        //Constructor to initialize the form
        public RoomInfo(IRoomInfoRepository repository)
        {
            InitializeComponent();
            _repository = repository;
        }

        //This method loads the room data into the DataGridView on form load
        private void RoomInfo_Load(object sender, EventArgs e)
        {
            try
            {
                DataTable roomData = _repository.GetAllRooms();
                dataGridView1.DataSource = new BindingSource(roomData, null);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error loading room data: " + ex.Message);
            }
        }

        //This method is triggered when the Save button is clicked to insert room info
        //into the database
        private void Button1_Click(object sender, EventArgs e)
        {
            try
            {
                //Validate the input fields (optional step, you can add more validation
                //logic as needed)
                if (string.IsNullOrWhiteSpace(textBox1.Text) ||
                    string.IsNullOrWhiteSpace(textBox2.Text) ||
                    string.IsNullOrWhiteSpace(textBox3.Text) ||
                    string.IsNullOrWhiteSpace(textBox4.Text) ||
                    string.IsNullOrWhiteSpace(textBox5.Text) ||
                    string.IsNullOrWhiteSpace(textBox6.Text))
                {
                    MessageBox.Show("Please fill all fields.");
                    return;
                }

                string building = textBox1.Text;
                string roomType = textBox2.Text;
                string roomNo = textBox3.Text;
                int bedCount = Convert.ToInt32(textBox4.Text);
            }
        }
    }
}

```

```

        decimal price = Convert.ToDecimal(textBox5.Text);
        string status = textBox6.Text;

        //Insert new room data into the database
        _repository.AddRoom(building, roomType, roomNo, bedCount, price, status);

        MessageBox.Show("Room information saved successfully.");

        //Clear input fields after successful save
        ClearFields();

        //Refresh the DataGridView to show the latest room data
        DataTable updatedRoomData = _repository.GetAllRooms();
        dataGridView1.DataSource = new BindingSource(updatedRoomData, null);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error saving room data: " + ex.Message);
    }
}


//This method is triggered when the Clear button is clicked to clear the input
fields private void Button2_Click(object sender, EventArgs e)
{
    ClearFields();
}

//Method to clear the input fields
private void ClearFields()
{
    textBox1.Clear();
    textBox2.Clear();
    textBox3.Clear();
    textBox4.Clear();
    textBox5.Clear();
    textBox6.Clear();
}
}
}

```

RoomInfo

Room Information



Building No

Room Type

Room No

No. Of Bed

Price

Status

Add

Reset

	Id	building	r_type	r_no	no_bed
▶	1	1	ghf	fgd	2
	2	2	rent	1	2
	3	3			
*					

<
>

Staff Information Form

```

using HospitalManagementSystemCSharp.Refactoring;
using System;
using System.Data;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class StaffInformation : Form
    {
        private readonly IStaffRepository _staffRepository;

        //Constructor to initialize the form
        public StaffInformation(IStaffRepository staffRepository)
        {
            InitializeComponent();
            _staffRepository = staffRepository;
        }
    }

```



```

//This method loads the staff data into the DataGridView on form load
private void StaffInformation_Load(object sender, EventArgs e)
{
    dataGridView1.DataSource = _staffRepository.GetAllStaff();
}

//This button will trigger the add staff method to insert the details of a new
staff member
private void Button1_Click(object sender, EventArgs e)
{
    string gender = radioButton1.Checked ? "Male" : "Female";
    try
    {
        _staffRepository.AddStaff(textBox2.Text, gender, textBox4.Text,
Convert.ToDecimal(textBox5.Text), textBox6.Text, textBox7.Text);
        MessageBox.Show("Staff Information Saved Successfully..");
        ClearFields();
        dataGridView1.DataSource = _staffRepository.GetAllStaff();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//This is a search button to search staff by ID
private void Button2_Click(object sender, EventArgs e)
{
    if (int.TryParse(textBox1.Text, out int id))
    {
        DataRow staff = _staffRepository.GetStaffById(id);
        if (staff != null)
        {
            textBox2.Text = staff["name"].ToString();
            radioButton1.Checked = staff["gender"].ToString() == "Male";
            radioButton2.Checked = !radioButton1.Checked;
            textBox4.Text = staff["position"].ToString();
            textBox5.Text = staff["salary"].ToString();
            textBox6.Text = staff["contact"].ToString();
            textBox7.Text = staff["address"].ToString();
        }
        else
        {
            MessageBox.Show($"Staff with ID {id} not found.");
            textBox1.Clear();
        }
    }
}

//This button triggers the update query to update staff info
private void Button4_Click(object sender, EventArgs e)
{
    string gender = radioButton1.Checked ? "Male" : "Female";
    try
    {

```

```

        _staffRepository.UpdateStaff(Convert.ToInt32(textBox1.Text),
textBox2.Text, gender, textBox4.Text, Convert.ToDecimal(textBox5.Text), textBox6.Text,
textBox7.Text);
        MessageBox.Show("Staff details updated successfully.");
        ClearFields();
        dataGridView1.DataSource = _staffRepository.GetAllStaff();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}


//Clear button
private void Button3_Click(object sender, EventArgs e)
{
    ClearFields();
}

//Method to clear the input fields
private void ClearFields()
{
    textBox1.Clear();
    textBox2.Clear();
    textBox4.Clear();
    textBox5.Clear();
    textBox6.Clear();
    textBox7.Clear();
    radioButton1.Checked = false;
    radioButton2.Checked = false;
}
}
}

```

StaffInformation

Staff Information



Staff Id

Name

Gender

☒ Male
 ☐ Female

Position

Salary

Contact

Address

Add

Search

Update

Reset

	Id	name	gender	position	salary
▶	1	doc	Female	senior	85000
	2	Yamir	Male	junior	85000
*					

< >

View Patient Checkout Form

```

using HospitalManagementSystemCSharp.Refactoring;
using System;
using System.Data;
using System.Windows.Forms;

namespace HospitalManagementSystemCSharp
{
    public partial class ViewPatientCheckOut : Form
    {
        private readonly ICheckoutRepository _checkoutRepository;
    }

```

```

//Constructor to initialize the form
public ViewPatientCheckOut(ICheckoutRepository checkoutRepository)
{
    InitializeComponent();
    _checkoutRepository = checkoutRepository;
}

//This method loads the checkout data into the DataGridView on form load
private void ViewPatientCheckOut_Load(object sender, EventArgs e)
{
    dataGridView1.DataSource = _checkoutRepository.GetAllCheckouts();
}

//This is a search button to search checkout of a patient by ID
private void Button1_Click(object sender, EventArgs e)
{
    if (int.TryParse(textBox1.Text, out int id))
    {
        dataGridView1.DataSource = _checkoutRepository.GetCheckoutById(id);
    }
    else
    {
        MessageBox.Show("Please enter a valid ID.");
    }
}
}
}

```

Patient Checkout

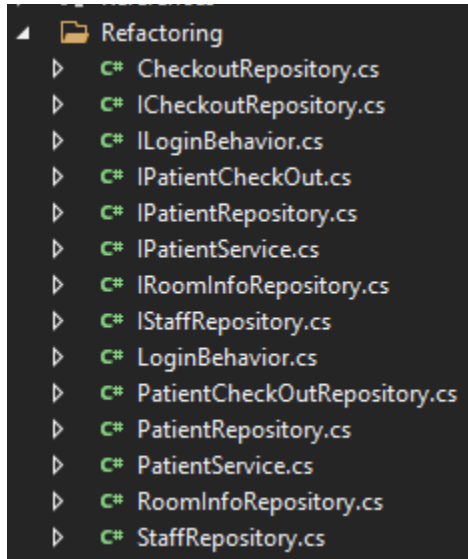
Checkout Id

Search

	Id	name	gen	age	contact
▶	11	Hamza	Male	21	1234567890
*					

Refactoring

The refactoring of this system includes addition of interfaces, methods, exception handlings and other necessary refactoring techniques. All refactoring classes and interfaces are stored inside a “Refactoring” folder.



CHAPTER # 5 TESTING

5.1 Testing Procedure & Test Cases:

For the testing procedure of our system, we will have the testing environment using the NUnit framework of C#. All the requirements and code optimizations are done. After tests, any tested data will no longer be available in the database and will be removed to avoid pollution in the system. Another point to note here is that if we apply tests for each module, then the purpose for refactoring will be disturbed, so we will apply tests on few modules to check most of the compatibility including the databases and the normal data logic. Following are some of the test cases we have used:

Login Functionality

To test the login functionality, we have the following code that will check the login by entering the correct and wrong data:

```
using NUnit.Framework;
using HospitalManagementSystemCSharp.Refactoring;

namespace HospitalManagementSystemCSharp.Tests
{
    [TestFixture]
    public class LoginBehaviorTests
    {
        private LoginBehavior _loginBehavior;

        [SetUp]
        public void Setup()
        {
            //Setup the Login form mock
            //Assuming you have a valid Login form to pass in
            var loginForm = new Login();
            _loginBehavior = new LoginBehavior(loginForm);
        }

        [Test]
        public void IsValidLogin_WithCorrectCredentials_ReturnsTrue()
        {
            //Inserting correct values
            var result = _loginBehavior.IsValidLogin("admin", "pass");
            Assert.IsTrue(result);
        }

        [Test]
        public void IsValidLogin_WithIncorrectCredentials_ReturnsFalse()
        {
            //Inseting wrong values
            var result = _loginBehavior.IsValidLogin("wronguser", "wrongpass");
        }
    }
}
```

```

        //return true, must be false with wrong credentials
        Assert.IsFalse(result);
    }
}

```

Patient Functionality

To test the patient functionality, we have the following code that will check the patient by entering a demo data into the database and then load it inside the data grids, if it succeeds, then it is ok and will be passed. It also includes the tests of database constraints validation so database integrity is prioritized:

```

using HospitalManagementSystemCSharp.Refactoring;
using NUnit.Framework;
using System;
using System.Data.SqlClient;

namespace HospitalManagementSystemCSharp.Tests
{
    [TestFixture]
    public class PatientCheckOutRepositoryTests
    {
        private PatientCheckOutRepository _repository;
        private string _connectionString = "Data Source=localhost\\SQLEXPRESS;Initial
Catalog=hms;Integrated Security=True";

        [SetUp]
        public void Setup()
        {
            _repository = new PatientCheckOutRepository();
            ClearTestDatabase();
        }

        [TearDown]
        public void TearDown()
        {
            ClearTestDatabase();
        }

        //Clears test data in the "checkout" and "patient" tables to ensure a clean test
        environment.
        private void ClearTestDatabase()
        {
            using (SqlConnection con = new SqlConnection(_connectionString))
            {
                con.Open();
                var clearCheckout = new SqlCommand("DELETE FROM checkout", con);
                clearCheckout.ExecuteNonQuery();

                var clearPatient = new SqlCommand("DELETE FROM patient", con);
                clearPatient.ExecuteNonQuery();
            }
        }
    }
}

```



```

        con.Close();
    }
}

[Test]
public void LoadPatientDetails_WithExistingPatientId_ReturnsCorrectData()
{
    //Insert a test patient, but do NOT insert the id manually
    int testPatientId;
    string expectedName = "John Doe";
    string expectedGen = "Male";
    string expectedAge = "30";
    string expectedContact = "123456789";
    string expectedAddr = "123 Street";
    string expectedDisease = "Flu";

    using (SqlConnection con = new SqlConnection(_connectionString))
    {
        con.Open();
        var insertPatientCmd = new SqlCommand(
            "INSERT INTO patient (name, gen, age, cont, addr, disease) " +
            "VALUES (@name, @gen, @age, @cont, @addr, @disease); " +
            "SELECT SCOPE_IDENTITY();", con); //SCOPE_IDENTITY returns the
generated identity value

        insertPatientCmd.Parameters.AddWithValue("@name", expectedName);
        insertPatientCmd.Parameters.AddWithValue("@gen", expectedGen);
        insertPatientCmd.Parameters.AddWithValue("@age", expectedAge);
        insertPatientCmd.Parameters.AddWithValue("@cont", expectedContact);
        insertPatientCmd.Parameters.AddWithValue("@addr", expectedAddr);
        insertPatientCmd.Parameters.AddWithValue("@disease", expectedDisease);

        //Get the auto-generated id of the inserted patient
        testPatientId = Convert.ToInt32(insertPatientCmd.ExecuteScalar());
        con.Close();
    }

    //Call LoadPatientDetails
    _repository.LoadPatientDetails(testPatientId, out string name, out string gen,
out string age, out string contact, out string addr, out string disease);

    //Assert checks
    Assert.AreEqual(expectedName, name);
    Assert.AreEqual(expectedGen, gen);
    Assert.AreEqual(expectedAge, age);
    Assert.AreEqual(expectedContact, contact);
    Assert.AreEqual(expectedAddr, addr);
    Assert.AreEqual(expectedDisease, disease);
}

[Test]
public void LoadPatientDetails_WithNonExistentPatientId_ReturnsEmptyStrings()
{
    //Call LoadPatientDetails with a non-existent patient id

```

```

        _repository.LoadPatientDetails(999, out string name, out string gen, out
string age, out string contact, out string addr, out string disease);

        //Assert checks
        Assert.AreEqual(string.Empty, name);
        Assert.AreEqual(string.Empty, gen);
        Assert.AreEqual(string.Empty, age);
        Assert.AreEqual(string.Empty, contact);
        Assert.AreEqual(string.Empty, addr);
        Assert.AreEqual(string.Empty, disease);
    }

    [Test]
    public void SavePatientCheckout_WithValidData_InsertsDataCorrectly()
    {
        //Test data for a patient checkout
        string name = "Jane Doe";
        string gen = "Female";
        string age = "28";
        string contact = "987654321";
        string addr = "456 Avenue";
        string disease = "Cold";
        string dateIn = "2024-12-20";
        string dateOut = "2024-12-21";
        string build = "Building A";
        string roomNo = "101";
        string roomType = "Single";
        string status = "Checked Out";
        string medPrice = "200";
        string total = "500";

        //Call SavePatientCheckout
        _repository.SavePatientCheckout(name, gen, age, contact, addr, disease,
dateIn, dateOut, build, roomNo, roomType, status, medPrice, total);

        //Verify that the record was actually inserted into the database
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            var selectCmd = new SqlCommand("SELECT COUNT(*) FROM checkout WHERE name =
@name AND contact = @contact", con);
            selectCmd.Parameters.AddWithValue("@name", name);
            selectCmd.Parameters.AddWithValue("@contact", contact);

            int count = (int)selectCmd.ExecuteScalar();
            con.Close();

            Assert.AreEqual(1, count, "The patient checkout record was not inserted
correctly.");
        }
    }
}

```

Checkout Functionality

To test the checkout functionality, we have the following code that will check the patient checkout by entering a demo data into the database and then load it inside the data grids, if it succeeds, then it is ok and will be passed. It also includes the tests of database constraints validation so database integrity is prioritized:

```
using NUnit.Framework;
using System;
using System.Data;
using System.Data.SqlClient;
using HospitalManagementSystemCSharp.Refactoring;

namespace HospitalManagementSystem.Tests
{
    [TestFixture]
    public class CheckoutRepositoryTests
    {
        private CheckoutRepository _checkoutRepository;
        private readonly string _connectionString = "Data
Source=localhost\\SQLEXPRESS;Initial Catalog=hms;Integrated Security=True";

        [SetUp]
        public void Setup()
        {
            //Initialize the class under test
            _checkoutRepository = new CheckoutRepository();
        }

        [Test]
        public void GetAllCheckouts_WhenCalled_ReturnsDataTable()
        {
            //Seed a test record in the checkout table
            int checkoutId;
            using (SqlConnection con = new SqlConnection(_connectionString))
            {
                con.Open();
                var insertCmd = new SqlCommand(
                    "INSERT INTO checkout (name, gen, age, contact, addr, disease,
date_in, date_out, build, r_no, r_type, status, med_price, total) " +
                    "VALUES (@name, @gen, @age, @contact, @addr, @disease, @date_in,
@date_out, @build, @r_no, @r_type, @status, @med_price, @total); " +
                    "SELECT SCOPE_IDENTITY();", con);

                insertCmd.Parameters.AddWithValue("@name", "Test Name");
                insertCmd.Parameters.AddWithValue("@gen", "Male");
                insertCmd.Parameters.AddWithValue("@age", "40");
                insertCmd.Parameters.AddWithValue("@contact", "123456789");
                insertCmd.Parameters.AddWithValue("@addr", "Test Address");
                insertCmd.Parameters.AddWithValue("@disease", "Test Disease");
                insertCmd.Parameters.AddWithValue("@date_in", DateTime.Now.ToString("yyyy-
MM-dd"));
            }
        }
    }
}
```

```

        insertCmd.Parameters.AddWithValue("@date_out",
DateTime.Now.AddDays(1).ToString("yyyy-MM-dd"));
        insertCmd.Parameters.AddWithValue("@build", "A");
        insertCmd.Parameters.AddWithValue("@r_no", "102");
        insertCmd.Parameters.AddWithValue("@r_type", "Single");
        insertCmd.Parameters.AddWithValue("@status", "Checked Out");
        insertCmd.Parameters.AddWithValue("@med_price", "100.00");
        insertCmd.Parameters.AddWithValue("@total", "200.00");

        //Get the generated ID of the inserted record
        checkoutId = Convert.ToInt32(insertCmd.ExecuteScalar());
        con.Close();
    }

    //Call GetAllCheckouts to fetch data
    DataTable result = _checkoutRepository.GetAllCheckouts();

    //Check if the result is not null and contains rows
    Assert.IsNotNull(result, "The result should not be null.");
    Assert.IsTrue(result.Rows.Count > 0, "The result should contain rows.");

    //Delete the test record from the checkout table
    using (SqlConnection con = new SqlConnection(_connectionString))
    {
        con.Open();
        var deleteCmd = new SqlCommand("DELETE FROM checkout WHERE id = @id",
con);

        deleteCmd.Parameters.AddWithValue("@id", checkoutId);
        deleteCmd.ExecuteNonQuery();
        con.Close();
    }
}

[Test]
public void GetCheckoutById_WhenCalledWithValidId_ReturnsDataTable()
{
    //Insert a test checkout record and get the auto-generated ID
    int checkoutId;
    using (SqlConnection con = new SqlConnection(_connectionString))
    {
        con.Open();
        var insertCmd = new SqlCommand(
            "INSERT INTO checkout (name, gen, age, contact, addr, disease,
date_in, date_out, build, r_no, r_type, status, med_price, total) " +
            "VALUES (@name, @gen, @age, @contact, @addr, @disease, @date_in,
@date_out, @build, @r_no, @r_type, @status, @med_price, @total); " +
            "SELECT SCOPE_IDENTITY();", con);

        insertCmd.Parameters.AddWithValue("@name", "John Doe");
        insertCmd.Parameters.AddWithValue("@gen", "Male");
        insertCmd.Parameters.AddWithValue("@age", "30");
        insertCmd.Parameters.AddWithValue("@contact", "123456789");
        insertCmd.Parameters.AddWithValue("@addr", "123 Street");
        insertCmd.Parameters.AddWithValue("@disease", "Flu");
        insertCmd.Parameters.AddWithValue("@date_in", "2024-12-15");
        insertCmd.Parameters.AddWithValue("@date_out", "2024-12-20");
    }
}

```

```

        insertCmd.Parameters.AddWithValue("@build", "A");
        insertCmd.Parameters.AddWithValue("@r_no", "101");
        insertCmd.Parameters.AddWithValue("@r_type", "Single");
        insertCmd.Parameters.AddWithValue("@status", "Checked Out");
        insertCmd.Parameters.AddWithValue("@med_price", "100.00");
        insertCmd.Parameters.AddWithValue("@total", "500.00");

        //Get the auto-generated checkout ID
        checkoutId = Convert.ToInt32(insertCmd.ExecuteScalar());
        con.Close();
    }

    //Call GetCheckoutById with the generated ID
    DataTable result = _checkoutRepository.GetCheckoutById(checkoutId);

    //Check if the result contains the expected data
    Assert.IsNotNull(result, "The result should not be null.");
    Assert.AreEqual(1, result.Rows.Count, "There should be exactly one row.");
    Assert.AreEqual(checkoutId, Convert.ToInt32(result.Rows[0]["id"]), "The ID
should match the inserted checkout ID.");

    //Delete the test data
    using (SqlConnection con = new SqlConnection(_connectionString))
    {
        con.Open();
        var deleteCmd = new SqlCommand("DELETE FROM checkout WHERE id = @id",
con);

        deleteCmd.Parameters.AddWithValue("@id", checkoutId);
        deleteCmd.ExecuteNonQuery();
        con.Close();
    }
}

[Test]
public void GetCheckoutById_WhenCalledWithInvalidId_ReturnsEmptyDataTable()
{
    //Generate an invalid ID (we assume 999999 does not exist)
    int invalidCheckoutId = 999999;

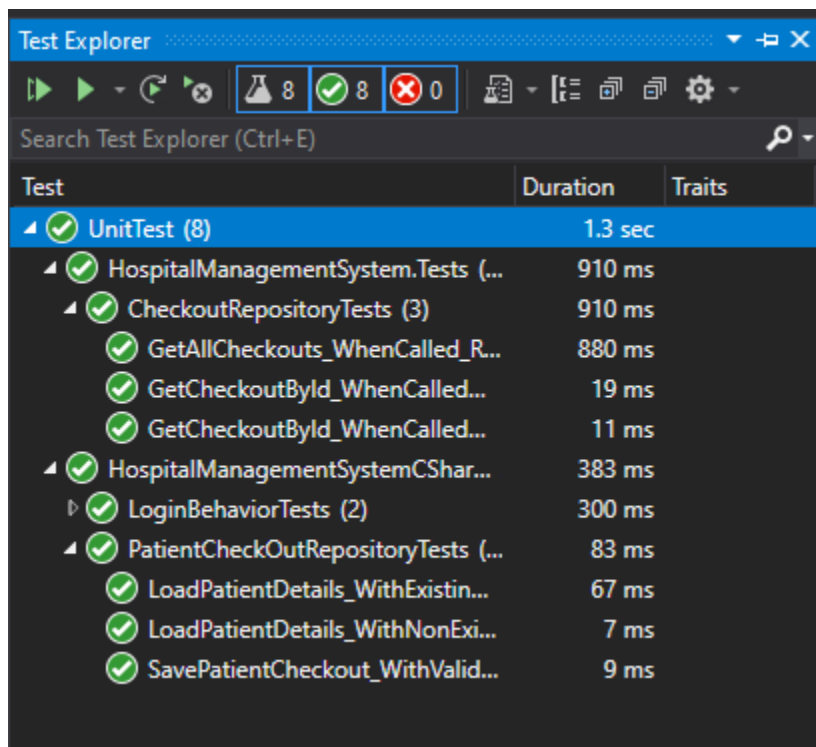
    //Call GetCheckoutById with the invalid ID
    DataTable result = _checkoutRepository.GetCheckoutById(invalidCheckoutId);

    //Check if the result is empty
    Assert.IsNotNull(result, "The result should not be null.");
    Assert.AreEqual(0, result.Rows.Count, "The result should be empty.");
}
}
}

```

5.2 Results

For our system, we have a separate project for unit tests. It utilizes the NUnit framework to test and validate methods within a project. NUnit is used to perform automatic testing for your logic, methods and classes. It supports features like assertions, test fixtures, test categories, and data-driven testing. Following are the results of each test case:



Test	Duration	Traits
✔ UnitTest (8)	1.3 sec	
✔ HospitalManagementSystem.Tests (...)	910 ms	
✔ CheckoutRepositoryTests (3)	910 ms	
✔ GetAllCheckouts_WhenCalled_R...	880 ms	
✔ GetCheckoutById_WhenCalled...	19 ms	
✔ GetCheckoutById_WhenCalled...	11 ms	
✔ HospitalManagementSystemCShar...	383 ms	
✔ LoginBehaviorTests (2)	300 ms	
✔ PatientCheckOutRepositoryTests (...)	83 ms	
✔ LoadPatientDetails_WithExistin...	67 ms	
✔ LoadPatientDetails_WithNonExi...	7 ms	
✔ SavePatientCheckout_WithValid...	9 ms	

This shows the efficiency and effectiveness of the methods and approach utilized to create this project. It also increases user trust and market adoption and applications.

CHAPTER # 6 FUTURE ENHANCEMENTS

6.1 Future Enhancements

Following are some future enhancements for our HMS:

Advanced Patient Management:

We can make a portal-based implementation of the system so the users can not only use it inside web but on mobile phones. Additionally, some AI integrations can be applied to automate some more processes.

Doctor & Staff Management:

We can make a different dashboards and user logins for the doctors and staff to be interactive with the system. They can make insights, personalize dashboards, check appointments etc.

Appointment Scheduling:

We can include AI features to automate the appointment process. Also, some SMS notifications and mailing reminders for appointments can be utilized. Use a portal to allocate appointments on mobile phone etc.

Billing & Payment Enhancements:

We can include the online payment integrations and multi-currency payment options. Auto billing features can be a very interesting addition with installment payments to remove any load of worries regarding delayed payment and money transfer issues.

CHAPTER # 7 APPENDIX

References

1. Hsieh, S. L., Lai, F., Cheng, P. H., Chen, J. L., Lee, H. H., Tsai, W. N., ... Chen, C. H. (2006). *An Integrated Healthcare Enterprise Information Portal and Healthcare Information System Framework. 2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. doi:10.1109/iembs.2006.260715

URL: <https://sci-hub.se/10.1109/iembs.2006.260715>
2. Qingzhang Chen, Jie Chen, Li, Y., & Fei Xu. (2010). *Design and implement of performance management system for hospital staff based on BSC. 2010 International Conference on Networking and Digital Society*. doi:10.1109/icnds.2010.5479261

URL: <https://sci-hub.se/10.1109/icnds.2010.5479261>
3. Yang, T. H., Cheng, P. H., Yang, C. H., Lai, F., Chen, C. L., Lee, H. H., ... Sun, Y. S. (2006). *A Scalable Multi-tier Architecture for the National Taiwan University Hospital Information System based on HL7 Standard. 19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*. doi:10.1109/cbms.2006.27

URL: <https://sci-hub.se/10.1109/cbms.2006.27>
4. Wangbin, Xieqi, Shihuaxin, Caoxinyu, Wangwenjing, & Chendi. (2014). *The design and implementation of inpatient medical expenses analysis system. 2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. doi:10.1109/bibm.2014.6999350

URL: <https://sci-hub.se/10.1109/bibm.2014.6999350>
5. Perdanakusuma, D., Puspitasari, W., & Saputra, M. (2020). *Utilizing Open ERP for Creating Medical Record Management System in Smart Hospital: A Case Study. 2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. doi:10.1109/iaict50021.2020.91720

URL: <https://sci-hub.se/10.1109/iaict50021.2020.91720>