Covid Bed Hospital Allotment

- Hari Narayanan M

B.Tech Information Technology,

Manipal Institute of Technology, Manipal

ABSTRACT

The pandemic brought about by Coronavirus has exposed that it is so imperative to oversee medical services assets actually, particularly with regards to assigning emergency clinic beds explicitly for Coronavirus patients. To address this issue, the Covid Hospital Bed Availability Database Project offers a full programming bundle that is planned to improve and facilitate bed distribution systems continuously in any pandemic like calamity. An outline of the undertaking's primary highlights, objectives, and expected impacts is given in this theoretical. This database task's fundamental objective is to make a reliable, versatile, and easy to use framework that gives precise and opportune data on clinic bed accessibility. The venture expects to minimize the chance of bed deficiencies, work on the dispersion of medical care assets, and give heads, patients, and medical care experts exact knowledge. Under the ACM Taxonomy, the basic role of this data set project lines up with the subcategories of Data Frameworks - Medical care, explicitly zeroing in on Wellbeing data frameworks. The framework plans to give convenient, exact, and open data to partners associated with medical care assets during the continuous pandemic. It is planned for different crowds, including medical clinic heads, medical care experts, patients, their families, government wellbeing offices, and the advancement group. The record follows the ACM Taxonomy by incorporating utilitarian and non-useful necessities, execution contemplations, wellbeing and safety efforts, programming quality credits, and basic business rules. Key functionalities incorporate constant information about the board, validation and approval components, reservation and delivery conventions, warning frameworks, and vigorous announcing abilities. The framework is intended to be solid, adaptable, and easy to use, underscoring information protection and security. References to industry best practices, medical care guidelines, and information security principles guarantee that the undertaking lines up with laid out rules inside the ACM Figuring Grouping Framework. By utilizing existing information, the Covid Hospital Bed Availability Database Project adds to the worldwide endeavors to battle another pandemic like the Coronavirus pandemic, offering a vital and innovatively progressed arrangement inside the ACM Taxonomy.

Table of Contents

Chapter 1
Introduction

The Covid Hospital Bed Availability Database Project was started in response to the Covid-19 pandemic's unparalleled demands on the healthcare system. The goal of this project is to provide real-time information on hospital beds available for Covid-19 patients through the use of an extensive software solution. Presenting a thorough overview of the project's development, features, and design considerations is the aim of this report.

This database project's main goal is to make it easier to manage hospital bed resources effectively so that if we face anything again like Covid-19 pandemic, we are prepared for it beforehand. The project intends to reduce the possibility of bed shortages, optimize the distribution of healthcare resources, and give administrators, patients, and healthcare professionals timely information by establishing a centralized and dynamic system.

The project consists of several elements intended to give healthcare facilities real-time bed tracking, quick reservation processes, and improve decision-making. The Covid Hospital Bed Availability Database is a comprehensive solution for efficiently managing healthcare resources because of its scope, which includes user identification, security protections, and strong reporting and logging methods.

This research makes use of data security standards, industry best practices, and pertinent healthcare laws to guarantee the creation of a reliable, legal, and user-focused software solution. The research aims to make a substantial contribution to the ongoing global efforts to prevent the next pandemic like Covid-19 by utilizing existing information and adhering to established rules.

Chapter 2
LITERATURE SURVEY

A number of studies have investigated various bed allocation strategies for COVID-19 patients. These strategies can be broadly categorized into two main approaches:

- Demand-driven approaches: These approaches allocate beds based on the real-time demand for beds. This can be done using a variety of methods, such as queuing theory, priority queuing, and simulation.
- Capacity-planning approaches: These approaches allocate beds based on the predicted demand for beds. This can be done using a variety of methods, such as historical data, forecasting models, and expert judgment.

The optimal bed allocation strategy for a particular hospital will depend on a number of factors, including the size of the hospital, the patient population, and the availability of resources.

Demand-driven approaches: Demand-driven approaches have the advantage of being responsive to real-time changes in demand. However, they can be difficult to implement in practice, as they require accurate and up-to-date information on patient demand.

Capacity-planning approaches: Capacity-planning approaches are easier to implement than demand-driven approaches, as they do not require real-time information on patient demand. However, they can be less responsive to changes in demand, which can lead to bed shortages or underutilization of beds.

Comparison of demand-driven and capacity-planning approaches:

A number of studies have compared demand-driven and capacity-planning approaches. These studies have found that demand-driven approaches can be more efficient than capacity-planning approaches, but they are also more complex and difficult to implement.

The optimal bed allocation strategy for a particular hospital will depend on a number of factors, including the size of the hospital, the patient population, and the availability of resources. Demand-driven approaches have the advantage of being responsive to real-time changes in demand, but they can be difficult to implement in practice. Capacity-planning approaches are easier to implement than demand-driven approaches, but they can be less responsive to changes in demand.

Additional considerations:

In addition to the factors mentioned above, there are a number of other considerations that should be taken into account when developing a bed allocation strategy for COVID-19 patients. These considerations include:

- The severity of illness of patients
- The availability of specialized care
- The need to minimize patient transfers
- The need to protect healthcare workers from infection

Chapter 3
OBJECTIVES/PROBLEM STATEMENT

The “Covid Bed Slot availability management” aims to handle the entire activity of allotting hospital beds to Covid patients. The software keeps track of all the information about the hospital beds that are occupied/vacant in the hospital, their availability, their complete details, and the total number of beds and patients available in the hospital. The user will find it easy in this automated system rather than using the manual writing system. The system contains a database where all the information will be stored safely. The system will be user-friendly and error-free.

Overview:

This project is helpful to track all the hospital beds and patient information and to allot the number of beds the patients wished to book. The software will be able to handle all the necessary information. The software to be produced is on the Covid Bed Slot availability management system. Here there are 2 users. They are the Patient and the receptionist. The first procedure is registering the people who are in need of using this application. Patients are allowed to log in or sign up if they have no account registered. It prompts them to see the availability of beds in different Hospitals. The patient can click on the available room and select the time period to stay after which they will pay the required amount to confirm their booking. Receptionists on the other hand can login which will prompt them to a UI to Add or Delete vacancies of beds. The user can update their password if they forget and also make multiple accounts. All the information will be entered into the system.

Scope:

- This project is helpful to track all the hospital beds and patient information and to allot the beds, the patients wished to book. The software will be able to handle all the necessary information.
- Physical implementation of the system on specific hardware devices.

Significance:

Efficient bed allocation plays a crucial role in managing healthcare resources during a pandemic, ensuring that patients receive the care they need when they need it. During the COVID-19 pandemic, hospitals worldwide faced unprecedented challenges due to the surge in patients requiring hospitalization. Bed allocation strategies became critical in determining how to distribute limited resources effectively while maintaining patient safety and quality of care.

Expected Outcome(s):

- Successful development and implementation of a functional Covid Bed Slot Allotment System meeting specified requirements.
- Enhanced capabilities for storing, retreivig, and managing patient data efficiently

Chapter 4

DATA DESIGN

ER Diagram

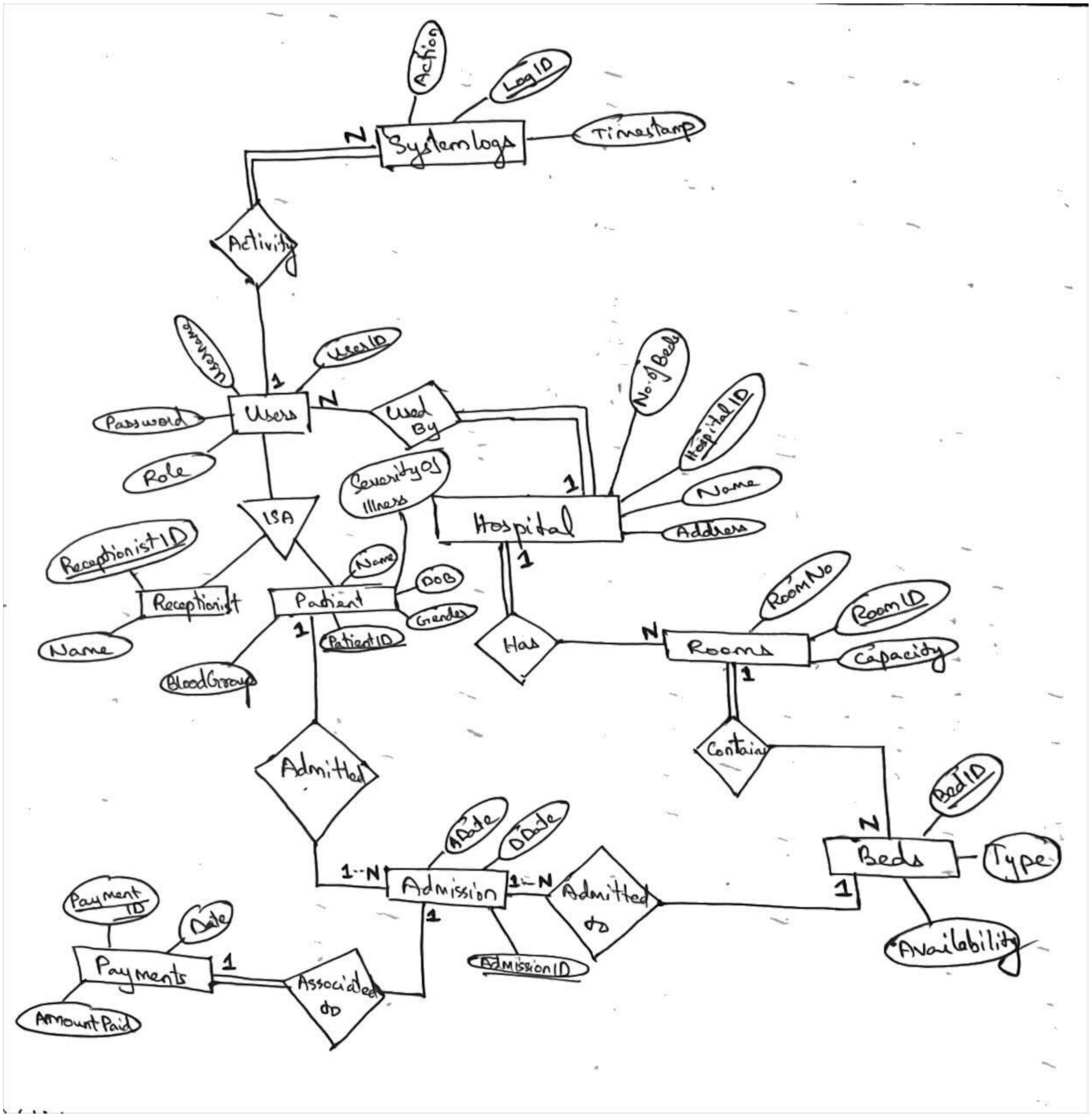


Fig1. ER Diagram

Reduction

The reduction phase in the database design process involves streamlining the complex Entity- Relationship (ER) diagram as given above in Fig. 1 into a simplified and focused representation. This reduction aims to emphasize essential components and relationships, enhancing clarity and understanding of the 'Covid Bed Slot Allotment' database structure. By condensing the ER diagram, we create a more manageable and concise view that highlights the core entities and their interconnections within the system. The following sections provide a reduced and refined presentation of the key database tables, offering a clearer insight into the structure of the project.

Tables:

a) Hospitals

- i) HospitalId - PK
- ii) HospitalName
- iii) Address
- iv) ContactNumber
- v) NumberOfBeds

b) Patients

- i) PatientId - PK
- ii) Name
- iii) DateOfBirth
- iv) Gender
- v) BloodGroup
- vi) SeverityOfIllness
- vii) UserId

c) Receptionist

- i) ReceptionId - PK
- ii) ReceptionistName
- iii) UserId

d) Rooms

- i) RoomId - PK
- ii) HospitalId
- iii) RoomNumber
- iv) Capacity

e) **Beds**
   i) BedId - PK
   ii) Type
   iii) RoomId
   iv) Availability

f) **Admissions**
   i) AdmissionId - PK
   ii) PatientId
   iii) AdmissionDate
   iv) DischargeDate
   v) BedId

g) **Payments**
   i. PaymentId - PK
   ii. AdmissionId
   iii. AmountPaid
   iv. PaymentDate

h) **Users**
   i) UserId - PK
   ii) Username
   iii) Password
   iv) Role

i) **SystemLogs**
   i) logId - PK
   ii) UserId
   iii) Action
   iv) Timestamp

j) **LoginReport**
   i) ID - PK
   ii) UserId
   iii) LoginTime

Schema:

Hospitals (**HospitalId**, HospitalName, Address, ContactNumber, NumberOfBeds)

Patients (**PatientId**, Name, DateOfBirth, Gender, BloodGroup, SeverityOfIllness, UserId)

Receptionist (**ReceptionistId**, ReceptionistName, UserId)

Rooms (**RoomId**, HospitalId, RoomNumber, Capacity)

Beds (**BedId**, Type, RoomId, Availability)

Admissions (**AdmissionId**, PatientId, AdmissionDate, DischargeDate, BedId)

Payments (**PaymentId**, AdmissionId, AmountPaid, PaymentDate)

Users (**UserId**, Username, Password, Role)

SystemLogs (**LogId**, UserId, Action, Timestamp)

LoginReport(**ID**,UserId,LoginTime)

Functional Dependencies

1) Hospitals:

(HospitalId) → (HospitalName, Address, ContactNumber, No. of Beds)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {HospitalId} is a superkey.

2) Users: (UserId) → (Username, Password, Role)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {UserId} is a superkey.

3) Patients: {PatientId} → {Name, DOB, Gender, BloodGroup, SeverityOfIllness,UserId}

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {PatientId} is a superkey.

4) Receptionists: {ReceptionId} → {ReceptionistName, UserId}

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {ReceptionId} is a superkey.

5) Rooms: (RoomId) → (HospidalId, RoomNo, Capacity)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {RoomId} is a superkey.

6)Beds: (BedId) → (Type, RoomId, Availability)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {BedId} is a superkey.

7) Admissions: (AdmissionId) → (Patient Id, Admission Dade, Discharge Date, BedId)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {AdmissionId} is a superkey.

8) Payments: (PaymentId) → (AdmissionId, AmoundPaid, PaymentDate)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {PaymentId} is a superkey.

9) SystemLogs : (Logld)→ (UserId, Action, Timestamp)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {Logld} is a superkey.

10) LoginReport: (ID)→(UserId, LoginTime)

- 1NF (First Normal Form): The table is already in 1NF because all attributes contain atomic values.
- 2NF (Second Normal Form): The table is already in 2NF because all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): The table is already in 3NF because there are no transitive dependencies.
- BCNF (Boyce-Codd Normal Form): The table is already in BCNF as the determinant {ID} is a superkey.

All the tables are BCNF form because, in their Functional dependencies, the LHS is the superkey in every case.

Normalization with Suitable Justification

To be in BCNF form, a table must satisfy two conditions:

- It must be in 3NF form.
- Every non-trivial functional dependency in the table must have a determinant that is a superkey.

A table is in 3NF form if it does not contain any partial or transitive dependencies. A partial dependency is a dependency where a non-prime attribute is functionally dependent on a part of the primary key. A transitive dependency is a dependency where one non-prime attribute is functionally dependent on another non-prime attribute, which is in turn functionally dependent on the primary key. To check whether a table is in BCNF form, we can identify all of the non-trivial functional dependencies in the table and then check whether the determinant of each dependency is a superkey. A superkey is a set of attributes that uniquely identifies each row in the table.

Creation of Tables:

```
CREATE TABLE `hospitals` (
 `HospitalId` int NOT NULL,
 `HospitalName` varchar(45) DEFAULT NULL,
 `Address` varchar(255) DEFAULT NULL,
 `ContactNumber` varchar(45) DEFAULT NULL,
 `NumberOfBeds` int DEFAULT NULL,
 PRIMARY KEY (`HospitalId`));


CREATE TABLE `users` (
 `UserId` int NOT NULL AUTO_INCREMENT,
 `Username` varchar(45) DEFAULT NULL,
 `Password` varchar(45) DEFAULT NULL,
 `Roles` varchar(15) DEFAULT NULL,
 `HospitalId` int DEFAULT NULL,
 PRIMARY KEY (`UserId`),
 KEY `HospitalId_idx` (`HospitalId`),
 CONSTRAINT `HospitalId` FOREIGN KEY (`HospitalId`) REFERENCES `hospitals` (`HospitalId`));

CREATE TABLE `patients` (
 `PatientId` int NOT NULL,
 `Name` varchar(45) DEFAULT NULL,
 `DateOfBirth` date DEFAULT NULL,
 `Gender` varchar(45) DEFAULT NULL,
 `BloodGroup` varchar(45) DEFAULT NULL,
 `SeverityOfIllness` varchar(45) DEFAULT NULL,
 `UserId` int DEFAULT NULL,
 PRIMARY KEY (`PatientId`),
 KEY `UserId_idx` (`UserId`),
 CONSTRAINT `UserId_Patients` FOREIGN KEY (`UserId`) REFERENCES `users` (`UserId`));

CREATE TABLE `receptionist` (
 `ReceptionistId` int NOT NULL,
 `ReceptionistName` varchar(45) DEFAULT NULL,
 `UserId` int DEFAULT NULL,
 PRIMARY KEY (`ReceptionistId`),
 KEY `UserId_idx` (`UserId`),
 CONSTRAINT `UserId_Receptionist` FOREIGN KEY (`UserId`) REFERENCES `users` (`UserId`));

CREATE TABLE `rooms` (
 `Roomid` int NOT NULL,
 `RoomNo` int DEFAULT NULL,
 `Capacity` int DEFAULT NULL,
 `HospitalId` int DEFAULT NULL,
 PRIMARY KEY (`Roomid`),
 KEY `HospitalId` (`HospitalId`),
 CONSTRAINT `rooms_ibfk_1` FOREIGN KEY (`HospitalId`) REFERENCES `hospitals` (`HospitalId`));
```

```
CREATE TABLE `beds` (
  `BedId` int NOT NULL,
  `Type` enum('General','ICU','Ventilator') DEFAULT NULL,
  `RoomId` int DEFAULT NULL,
  `Availability` enum('Available','Occupied') DEFAULT NULL,
  PRIMARY KEY (`BedId`),
  KEY `RoomId_idx` (`RoomId`),
  CONSTRAINT `RoomId` FOREIGN KEY (`RoomId`) REFERENCES `rooms` (`Roomid`));

CREATE TABLE `admissions` (
  `AdmissionsId` int NOT NULL,
  `PatientId` int DEFAULT NULL,
  `AdmissionDate` date DEFAULT NULL,
  `DischargeDate` date DEFAULT NULL,
  `BedId` int NOT NULL,
  PRIMARY KEY (`AdmissionsId`),
  KEY `PatientId` (`PatientId`),
  KEY `BedId` (`BedId`),
  CONSTRAINT `admissions_ibfk_1` FOREIGN KEY (`PatientId`) REFERENCES `patients` (`PatientId`),
  CONSTRAINT `admissions_ibfk_2` FOREIGN KEY (`BedId`) REFERENCES `beds` (`BedId`)) ;

CREATE TABLE `payments` (
  `PaymentId` int NOT NULL AUTO_INCREMENT,
  `AdmissionsId` int DEFAULT NULL,
  `AmountPaid` varchar(45) DEFAULT NULL,
  `PaymentDate` date DEFAULT NULL,
  `AccountNo` varchar(45) DEFAULT NULL,
  `IFSC` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`PaymentId`),
  KEY `AdmissionsId_idx` (`AdmissionsId`),
  CONSTRAINT `AdmissionsId` FOREIGN KEY (`AdmissionsId`) REFERENCES `admissions` (`AdmissionsId`));
```

```
CREATE TABLE `login_report` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `UserID` int DEFAULT NULL,
  `LoginTime` datetime DEFAULT NULL,
  PRIMARY KEY (`ID`),
  KEY `Userid_login_idx` (`UserID`),
  CONSTRAINT `Userid_login` FOREIGN KEY (`UserID`) REFERENCES `users` (`UserId`));

CREATE TABLE `systemlogs` (
  `LogId` int NOT NULL AUTO_INCREMENT,
  `UserId` int DEFAULT NULL,
  `Action` varchar(145) DEFAULT NULL,
  `Timestamp` datetime DEFAULT NULL,
  PRIMARY KEY (`LogId`),
  KEY `Userid_log_idx` (`UserId`),
  CONSTRAINT `Userid_log` FOREIGN KEY (`UserId`) REFERENCES `users` (`UserId`)
);
```

Triggers

There are 5 triggers that are used in this project. Each one to insert into Systemlogs table whenever a user signs up, logs in, updates his/her password, does payment for a bed (incase of patient), adds a bed for vacancy (in case of receptionist), resprctively.

1) <u>If a new user signs up this trigger will fire:</u>

```
CREATE TRIGGER users_AFTER_INSERT AFTER INSERT ON users
FOR EACH ROW
BEGIN
        DECLARE action_text VARCHAR(45);
        IF (SELECT COUNT(*) FROM Users WHERE UserId = NEW.UserId) = 1 THEN
    SET action_text = 'User signed up';
    ELSE
        SET action_text = 'No Action';
    END IF;

    INSERT INTO systemlogs (UserId, Action, Timestamp)
    VALUES (NEW.UserId, action_text, NOW());
END
```

2) If a user updates their password:

```
CREATE TRIGGER users_AFTER_UPDATE AFTER UPDATE ON users
FOR EACH ROW
BEGIN
        DECLARE action_text VARCHAR(45);
    IF OLD.Password <> NEW.Password AND OLD.Username = NEW.Username THEN
        SET action_text = 'Password updated';
        INSERT INTO systemlogs (UserId, Action, Timestamp)
        VALUES (NEW.UserId, action_text, NOW());
    END IF;
END
```

3) If a user logs in:

```
CREATE TRIGGER login_report_AFTER_INSERT AFTER INSERT ON login_report
FOR EACH ROW
BEGIN
        DECLARE action_text VARCHAR(45);
    IF EXISTS (SELECT 1 FROM Users WHERE UserId = NEW.UserId) THEN
        SET action_text = 'User logged in';
    ELSE
        SET action_text = 'No Action';
    END IF;
    INSERT INTO systemlogs (UserId, Action, Timestamp)
    VALUES (NEW.UserId, action_text, NOW());
END
```

4) If a patient books a bed:

```
CREATE TRIGGER admissions_AFTER_INSERT AFTER INSERT ON admissions
FOR EACH ROW
BEGIN
        DECLARE action_text VARCHAR(200);
    DECLARE user_id INT;

    SELECT P.UserId INTO user_id
    FROM Patients P
    WHERE P.PatientId = NEW.PatientId;

        SET action_text = CONCAT('User ', user_id, ' Booked a bed under PatientId ', NEW.PatientId, ' on
AdmissionId ', NEW.AdmissionsId);

    INSERT INTO systemlogs (UserId, Action, Timestamp)
    VALUES (user_id, action_text, NOW());
END
```

5) If a receptionist add vacancies for beds:

```
CREATE TRIGGER beds_AFTER_INSERT AFTER INSERT ON beds
FOR EACH ROW
BEGIN
        DECLARE action_text VARCHAR(200);
    DECLARE user_id INT;
    SELECT U.UserId INTO user_id
    FROM users U
    INNER JOIN rooms R ON R.RoomId = NEW.RoomId
    INNER JOIN hospitals H ON R.HospitalId = H.HospitalId
    WHERE U.HospitalId = H.HospitalId
    LIMIT 1;

    SET action_text = CONCAT('User ', user_id, ' inserted a new ', NEW.Type, ' bed in Room ', NEW.RoomId);

    INSERT INTO systemlogs (UserId, Action, Timestamp)
    VALUES (user_id, action_text, NOW());
END
```

Chapter 5
METHODOLOGY

Implementation

1. Technology Stack

Programming Language: This project is developed using the C# programming language, chosen for its versatility, readability, and a wide range of libraries and frameworks suitable for web development.

Database Management System (DBMS): MySQL is employed as the backend database management system. Its relational structure facilitates efficient data storage and retrieval, crucial for managing book details, user information, and transaction records.

Graphical User Interface (GUI) Library: The GUI is created using Visual C# by Visual Studio Microsoft. This application provides a user-friendly interface for both customers and receptionists, ensuring a seamless and intuitive browsing experience.

2. System Architecture

Overview: This project adopts a simplified architecture where the primary interaction occurs locally on the user's machine. Unlike a conventional client- server architecture, it relies on a standalone application approach where the user interface, data processing, and storage are handled locally without the need for continuous communication with a remote server.

Local Application Architecture: In this design, patientss and receptionists interact directly with the system through a local Visual C# user interface. The system processes data, manages inventory, and handles user interactions entirely on the user's machine. This approach provides a user-friendly and self-contained experience, eliminating the need for continuous internet connectivity.

Key Features:

1. Local Processing: All data processing, including order placement, inventory management, and reporting, is conducted locally on the user's device.
2. Responsive User Interface: The tkinter-based interface ensures a responsive and seamless user experience, allowing customers and administrators to navigate the system effortlessly.
3. Data Storage: The MySQL database is utilized for local data storage, enabling efficient retrieval and management of information without reliance on external servers.

Benefits of Localized Architecture:

1. Offline Accessibility: Users can interact with OBMS even in the absence of an internet connection, promoting accessibility in various environments.
2. Reduced Latency: Local processing minimizes latency, enhancing the responsiveness of the system for a smoother user experience.

3. <u>Customer-Focused Features</u>

1. User Registration and Login: Customers register by providing essential information, and the system validates and stores this data securely. The login system ensures secure access control, protecting user accounts from unauthorized access.
2. Order Placement and Tracking: Customers can browse available books, place orders, and track the status of their purchases. Real-time updates are provided on the processing and delivery stages of the order.

4. <u>Administrator Features</u>

- Inventory Management: Receptionists can add or delete, bed details through an intuitive interface. The system keeps track of inventory levels and updates in real-time.
- Order Tracking and Reporting: A comprehensive order tracking system allows administrators to monitor order details, and reporting features provide insights into transaction histories for informed decision-making.

5. <u>Security Measures</u>

- User Authentication: User authentication is designed to ensure local data security. While robust encryption algorithms are typically employed in networked applications, in a local VC# interface, the emphasis is on protecting user credentials stored on the user's machine.
- Data Storage Security: Sensitive information, including user credentials and financial transactions, is stored locally on the user's device. Since the application operates in an isolated environment without network communication, the emphasis is on securing data at rest.

  Industry-standard encryption methods may still be applied to protect stored data integrity and user privacy within the confines of the local machine.

  Key Considerations:

1. Local Environment Focus: The security measures in this project are tailored to the local environment, prioritizing the protection of data stored on the user's machine.
2. No Network Transmission: Since covid bed slot allotment system does not involve the transmission of sensitive data over networks or external servers, encryption measures are adapted to secure data within the local application.

<u>Benefits of Localized Security:</u>

1. Reduced Complexity: Localized security measures streamline the application's architecture, focusing on safeguarding data within the local environment.
2. User Privacy: Protecting user credentials and sensitive information stored locally contributes to maintaining user privacy and data integrity.

Block Diagram

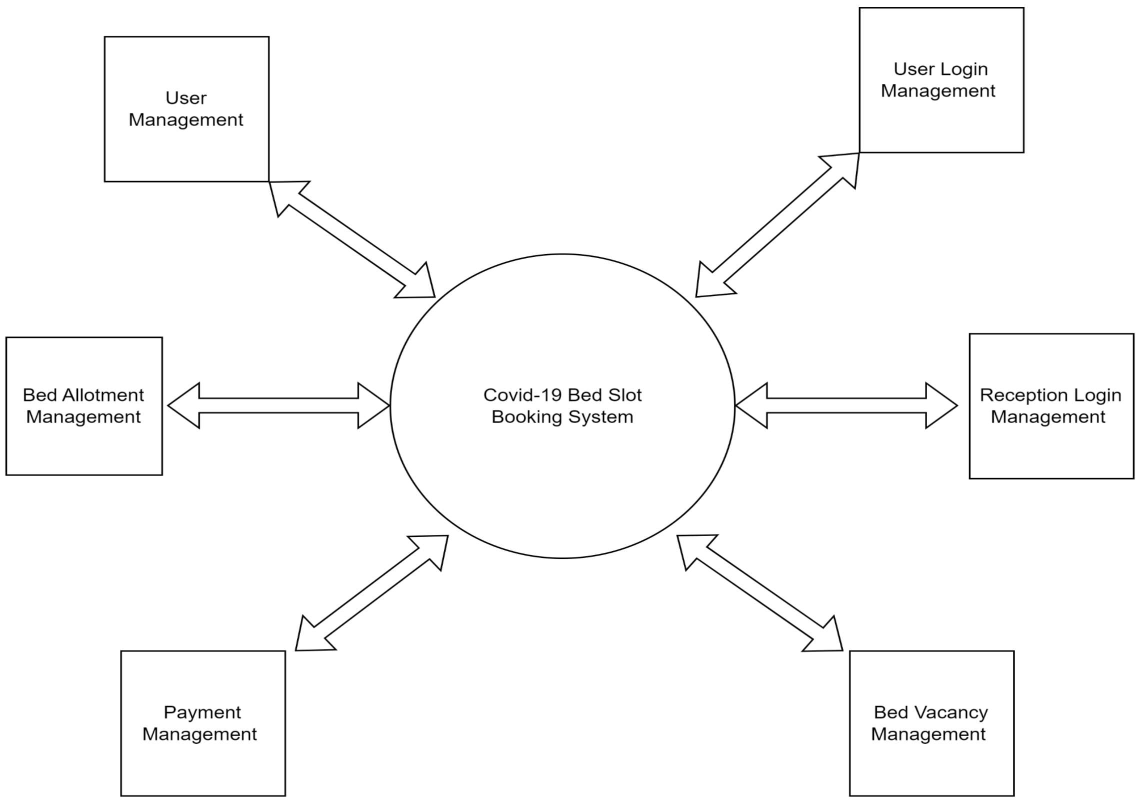


Fig 2. Block Diagram

Chapter 6

RESULTS

The implementation of the 'Covid-19 Bed Slot Allotment System' has demonstrated a successful integration of key features and functionalities. Through thorough testing and user engagement, the system has proven to be a reliable and user-friendly platform for managing patient and hospital operations online. Users have provided positive feedback, emphasizing the system's simplicity and effectiveness in book browsing, ordering, and account management.

In addition to user interactions, the administrative functionalities, including inventory management and payment, have been effectively implemented. The system's real-time data processing capabilities contribute to improved decision-making for Hospital administrators. The results affirm that this hospital fulfills its objectives, providing a practical solution for online hospital management. The project's success highlights the potential impact of implementing technology to enhance traditional processes.

Patient Interface:

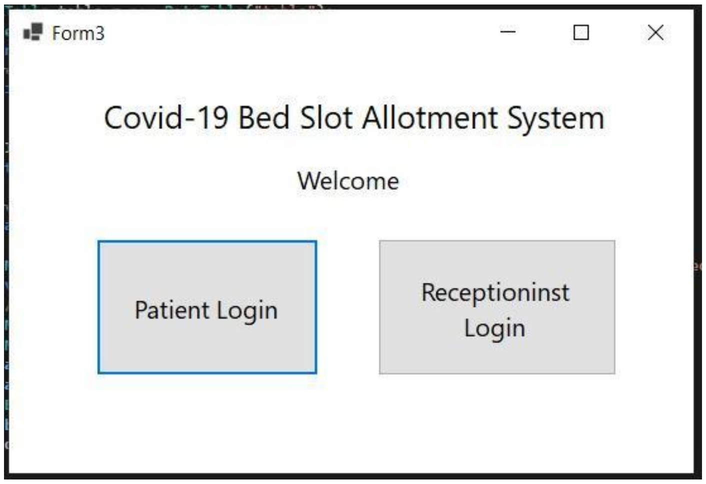
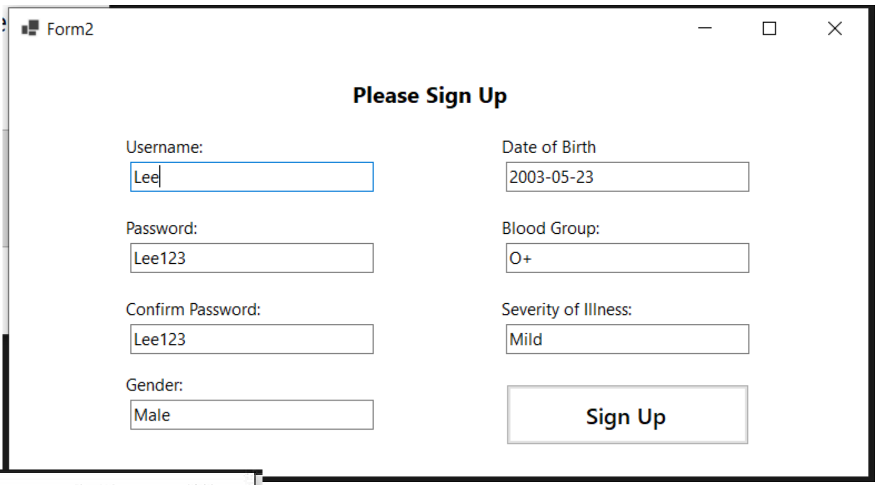


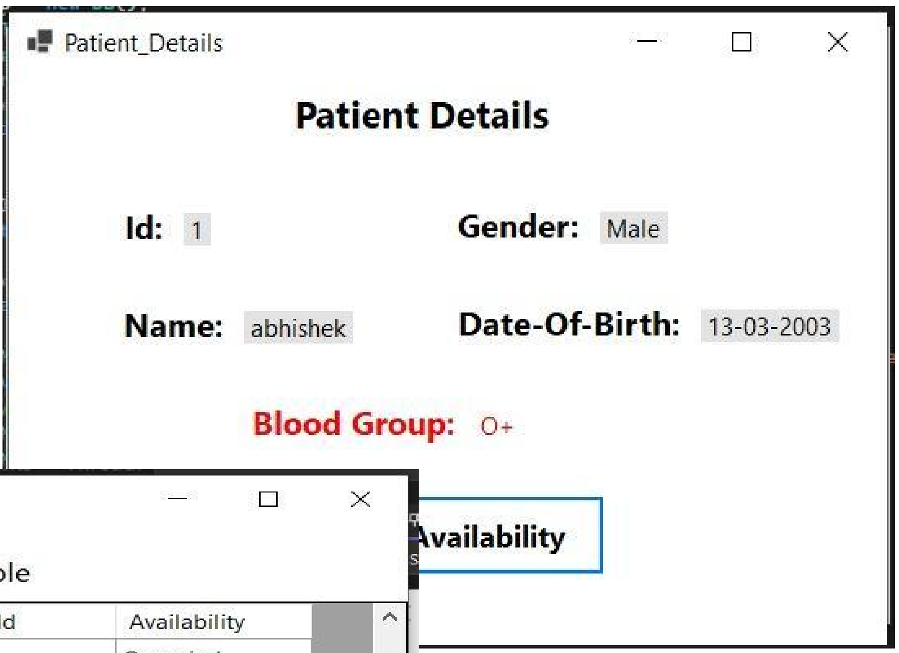
Fig 3. Opening Page



Username: Lee	Date of Birth 2003-05-23
Password: Lee123	Blood Group: O+
Confirm Password: Lee123	Severity of Illness: Mild
Gender: Male	

Fig 4. User Sign Up Page

Fig 5. Patient Log In Page

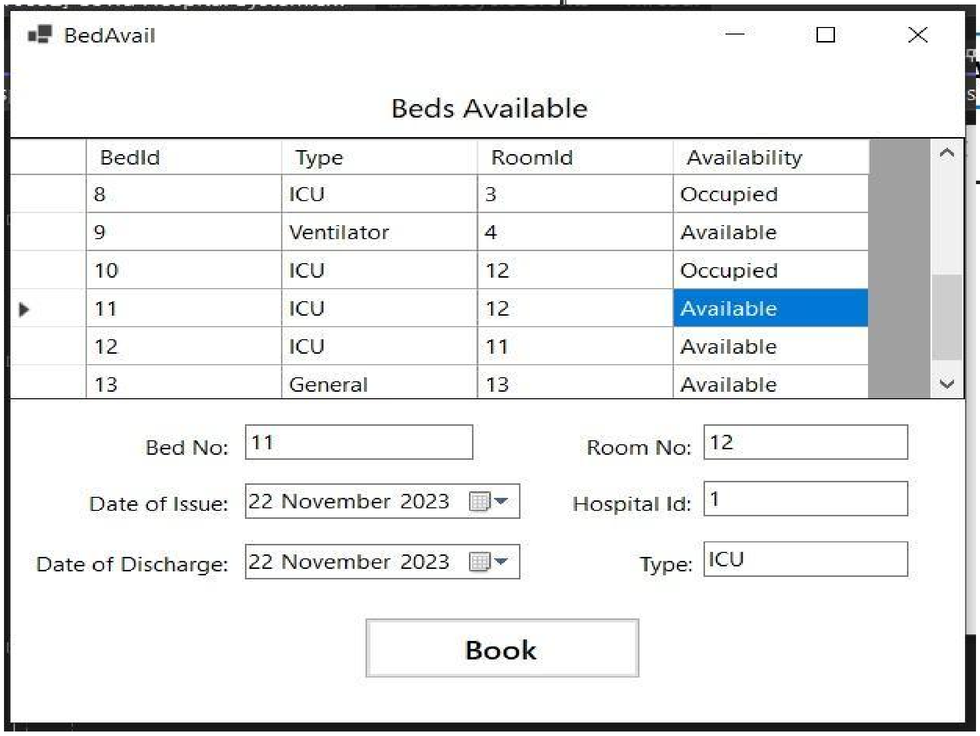


Patient_Details

Patient Details

Id:	1	**Gender:**	Male
Name:	abhishek	**Date-Of-Birth:**	13-03-2003
Blood Group:	O+		

Fig 6. Patient Details Page



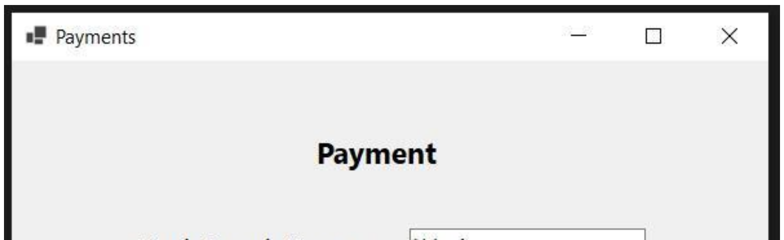
BedAvail

Beds Available

BedId	Type	RoomId	Availability
8	ICU	3	Occupied
9	Ventilator	4	Available
10	ICU	12	Occupied
11	ICU	12	Available
12	ICU	11	Available
13	General	13	Available

Bed No:	11	Room No:	12
Date of Issue:	22 November 2023	Hospital Id:	1
Date of Discharge:	22 November 2023	Type:	ICU

Fig 7. Show Bed Availability Page

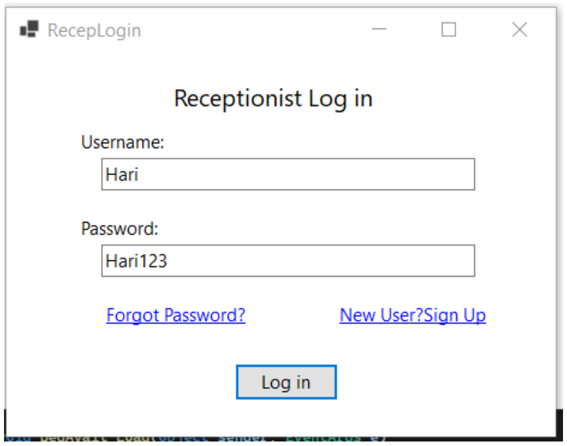


Payments

Payment

Fig 8. Payments page

Receptionist Interface:

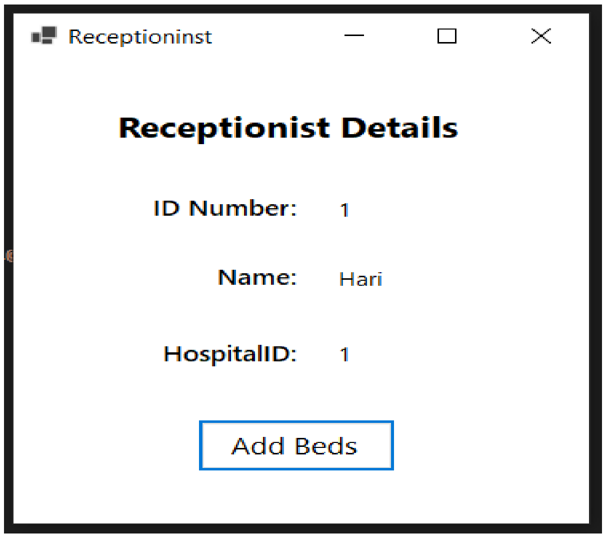


ReceptLogin

Receptionist Log in

Username: Hari

Password: Hari123



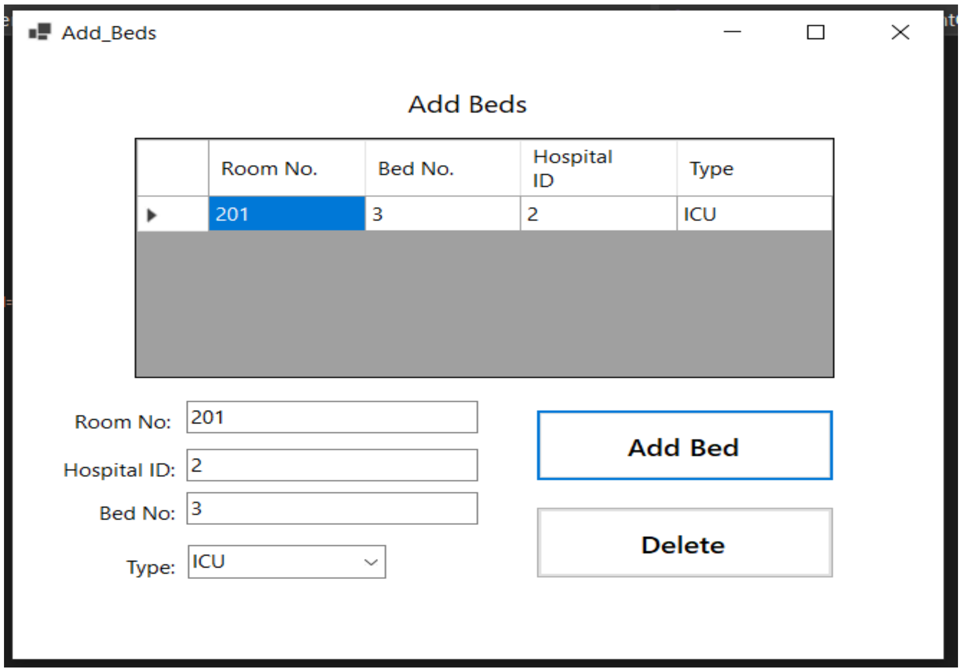
Receptioninst

Receptionist Details

ID Number: 1

Name: Hari

HospitalID: 1



Add_Beds

Add Beds

	Room No.	Bed No.	Hospital ID	Type
▶	201	3	2	ICU

Room No: 201

Hospital ID: 2

Bed No: 3

Type: ICU

Fig 9. (a)Receptionist Login Page (b) Details Page (c) Add Vacancies for Bed Page

Chapter 7

CONCLUSION

All in all, the Covid Hospital Bed Availability Database Project means an essential step towards strengthening medical services frameworks against upcoming pandemics like Coronavirus. This solution, including real-time bed tracking addresses current difficulties as well as lays out a strong system for future emergencies. We hope that our cooperative endeavors may bring about an ease in bed allotment and it lines up with industry guidelines and medical care guidelines. As we finish up this task, it is apparent that the examples learned will prepare for progressing development and improvement. While shutting this section, we aimed at providing medical services flexibility in one way. Consistent assessment and joint effort with medical care specialists and technicians to maintain the database will be significant, guaranteeing the framework stays at the front line of pandemic-like situations.

Future Scope:

While this project has achieved its primary objectives, there is considerable potential for further enhancements and expansions. The future scope of includes:

1. Mobile Application Development: Exploring the development of a mobile application to extend accessibility and cater to users who prefer mobile platforms.

2. Enhanced Reporting and Analytics: Introducing advanced reporting features and analytics tools to provide administrators with deeper insights into sales trends, customer behaviors, and inventory performance.

3. Implementing Machine Learning for Recommendations: Leveraging machine learning algorithms to analyze user preferences and offer personalized book recommendations, creating a more tailored user experience.

4. Geographic Expansion: Expanding the reach of this project to serve a broader geographic audience, considering localization and multi-language support.

5. Gamification for User Engagement: Introducing gamification elements to incentivize user engagement, such as loyalty programs, rewards, or challenges tied to book exploration.

6. Continuous Security Audits: Conducting regular security audits to adapt to evolving cybersecurity threats and ensure the ongoing protection of user data.

In conclusion, Covid-19 Bed Slot Management System has laid a strong foundation for an innovative and efficient online hospital management system. Its success opens the door to future developments and improvements, ensuring that it remains at the forefront of meeting the dynamic needs of medical care in the ever-evolving digital landscape. The journey of this project continues, driven by a commitment to excellence and a vision for the future of online bookstores.

8. References

1. **https://www.hindawi.com/journals/mpe/2020/8198563/**

2. **https://www.researchgate.net/figure/Predicted-number-of-beds-needed-and-ICU-beds-needed-in-Lombardy-Italy_fig2_351445096**
3. **https://thorax.bmj.com/content/76/3/302**

Appendix

Glossary:

BCNF (Boyce-Codd Normal Form): A specific level of normalization in the database design process, ensuring that certain dependencies are satisfied.

ER Diagram (Entity-Relationship Diagram): A visual representation of the entities and their relationships within the OBMS database.

GUI (Graphical User Interface): The visual interface that allows users to interact with the OBMS system using graphical elements.

MySQL: A relational database management system used as the backend database for storing and retrieving data in Covid Hospital Bed Availability Database.

Normalization: The process of organizing data in the database to reduce redundancy and improve data integrity.

Offline Accessibility: The capability of users to interact with OBMS even in the absence of an internet connection.

User Authentication: The process of verifying the identity of users to ensure secure access control within the Bed Allotment system.

Visual Studio C#: an integrated development environment (IDE) and programming language developed by Microsoft for developing applications for Windows, macOS, Linux, iOS, and Android. It is a popular choice for developing a wide variety of applications, including web applications, desktop applications, and mobile applications.