

Event Registration with Management System

A MINI-PROJECT BY:

Kabilesh P 230701133

Kamalesh P 230701137

in partial fulfillment of the award of the degree

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project report “**Event Registration with Management System**” is a Bonafide work of “KABILESH P (230701133) AND KAMALESH.P (230701137)” .

Submitted for the Practical Examination held on _____

Signature

Ms. Dharshini B S

Assistant Professor,

Computer Science and Engineering,

Rajalakshmi Engineering College (Autonomous),

Thandalam, Chennai-602 105

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to everyone who has contributed to the successful completion of this mini project.

First and foremost, I am deeply thankful to my Professor **Ms. Dharshini B S** my project advisor, for their invaluable guidance, insightful feedback, and continuous support throughout the duration of this mini project. Their expertise and encouragement have been instrumental in shaping my research and bringing this mini project to completion.

I would also like to express my appreciation to the faculty and staff of the **Computer Science and Engineering Department** at Rajalakshmi Engineering College for providing the necessary resources and a conducive learning environment. We express our sincere thanks to **Dr. P. Kumar, M.E., Ph.D.**, Professor and Head of the Department Computer Science and Engineering for his guidance and encouragement throughout the project work.

My heartfelt thanks go to my peers and friends for their collaboration, constructive criticism, and moral support.

Thank you all for your contributions, both direct and indirect, to the success of this project.

Abstract of the Project :

The Event Management System is a desktop application developed in Java using the JavaFX framework for the frontend and MySQL for the backend. This project aims to provide an efficient and user-friendly platform to manage various aspects of event organization, including event registration, billing, payments, and college collaborations. By integrating modern database handling techniques, the system ensures accurate data management and seamless user interaction.

The system is divided into multiple functional modules, each designed to handle a specific aspect of event management:

- 1. Home Page: Serves as the main navigation menu, allowing users to access different modules, such as events, event registration, billing, payments, college information, and user profiles.**
- 2. Event Module: Enables the creation, listing, and management of events, including event details and schedules.**
- 3. Event Registration Module: Facilitates participant registration, ensuring all necessary information is captured in the database.**
- 4. Billing Module: Handles payment generation and record-keeping for participant fees or other financial transactions.**
- 5. Payments Module: Displays detailed payment records, including user IDs, payment amounts, and payment dates, ensuring transparency and accountability.**
- 6. College Module: Manages information about collaborating colleges, such as names, locations, and years of establishment.**
- 7. Profile Module: Allows users to view and update their profiles for personalization.**

This project demonstrates the practical application of software engineering principles, including modular design, database management, and user interface development. The system is scalable and can be expanded to include additional features, such as reports, analytics, or integrations with

external APIs, making it an ideal solution for small- to medium-scale event management.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Introduction

1.2 Objectives

1.3 Modules

2. SURVEY OF TECHNOLOGIES

2.1 Software Description

2.2 Languages

2.3 SQL

2.4 Java

3. REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

3.2 Hardware and Software Requirements

3.3 Architecture Diagram

3.4 ER Diagram

3.5 Normalization

4. DESIGN AND IMPLEMENTATION

4.1 Database Design

5. SOURCE CODE

6. SNAPSHOTS

7. CONCLUSION

8. REFERENCES

1. INTRODUCTION

1.1 Introduction

The **Database Management System (DBMS)** project focuses on creating a structured and efficient way to manage data for various applications. It ensures data integrity, reduces redundancy, and provides an interface for performing CRUD (Create, Read, Update, Delete) operations. The project demonstrates the integration of a relational database with a programming language to develop a functional system that meets specific user requirements.

1.2 Objectives

- To design and implement a relational database for efficient data storage and retrieval.
- To ensure data integrity and consistency through normalization.
- To provide user-friendly interfaces for performing operations on the database.
- To integrate SQL with a programming language for seamless database interaction.

1.3 Modules

1. **User Authentication:** Secure login and access control.
2. **Data Management:** CRUD operations on the database.
3. **Reports:** Generate reports from the stored data.
4. **Validation:** Ensure data accuracy and security.
5. **Backup and Restore:** Facilitate database maintenance.

Supporting potential future enhancements, such as integrating insurance claims or online payment systems.

C2. SURVEY OF TECHNOLOGIES

2.1 Software Description

The project utilizes **MySQL** for the database, renowned for its robustness, scalability, and wide usage in both small and enterprise-level applications. It provides reliable data storage and supports complex queries with ease.

Java is chosen as the programming language due to its flexibility, platform independence, and strong community support. Java facilitates both backend development and frontend design, making it an ideal choice for building a comprehensive system with a user-friendly interface and seamless integration with the MySQL database.

2.2 Languages

The system is built using the following languages:

- **SQL (Structured Query Language):** SQL is the core language used for managing and manipulating relational databases. It allows us to create, read, update, and delete records within the MySQL database. SQL is crucial for implementing the data operations and performing complex queries for reporting and data retrieval.
- **Java:** Java is the primary language used for backend logic and GUI development. It connects seamlessly with MySQL using JDBC (Java Database Connectivity) for querying and updating the database. The language's object-oriented nature also aids in organizing the system's functionality effectively. Java is also utilized for the creation of a rich, interactive user interface (UI), making the application intuitive and easy to use.

2.3 SQL

SQL is employed in this project for the efficient design, manipulation, and querying of the relational database. It provides a standardized way of

interacting with the database to retrieve, modify, and manage the stored data. The SQL queries handle operations such as:

- **Data Retrieval:** SELECT queries are used to fetch specific patient, billing, and service-related information.
- **Data Insertion:** INSERT queries are used to add new records for patients, services, medications, and payments.
- **Data Modification:** UPDATE queries allow modification of patient details or billing information as required.
- **Data Deletion:** DELETE queries are used to remove records from the database when necessary.

The use of SQL ensures efficient management of the database, optimizing performance while maintaining data integrity and consistency.

2.4 Java

Java plays a significant role in the development of both the backend logic and the user interface of this project. As an object-oriented language, it allows for a modular and maintainable approach to coding, ensuring that each component of the system is self-contained and interacts with others in an organized manner.

- **Backend Logic:** Java handles the business logic, ensuring that data from the database is processed correctly and efficiently. It manages interactions with the MySQL database through JDBC to execute SQL queries, retrieve data, and update records.
- **Database Connectivity:** Using JDBC, Java establishes a connection to the MySQL database, allowing for seamless communication between the frontend and the database. This ensures that user actions in the GUI are reflected in the database in real-time.
- **User Interface (UI):** Java, using libraries like JavaFX or Swing, is used to build the graphical user interface. The UI is designed to be simple, interactive, and intuitive, providing an easy way for users to interact with the system, view patient details, generate bills, and manage payments.

Together, MySQL and Java offer a strong foundation for developing a reliable, scalable, and user-friendly hospital management system.

3. REQUIREMENTS AND ANALYSIS

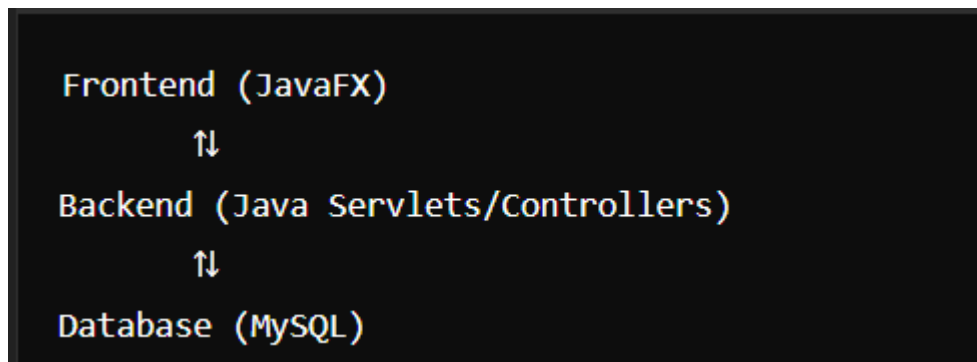
3.1 Requirement Specification

- **Functional Requirements:**
 - User authentication.
 - CRUD operations for data.
 - Report generation.
- **Non-Functional Requirements:**
 - Scalability to handle large datasets.
 - High performance and reliability.

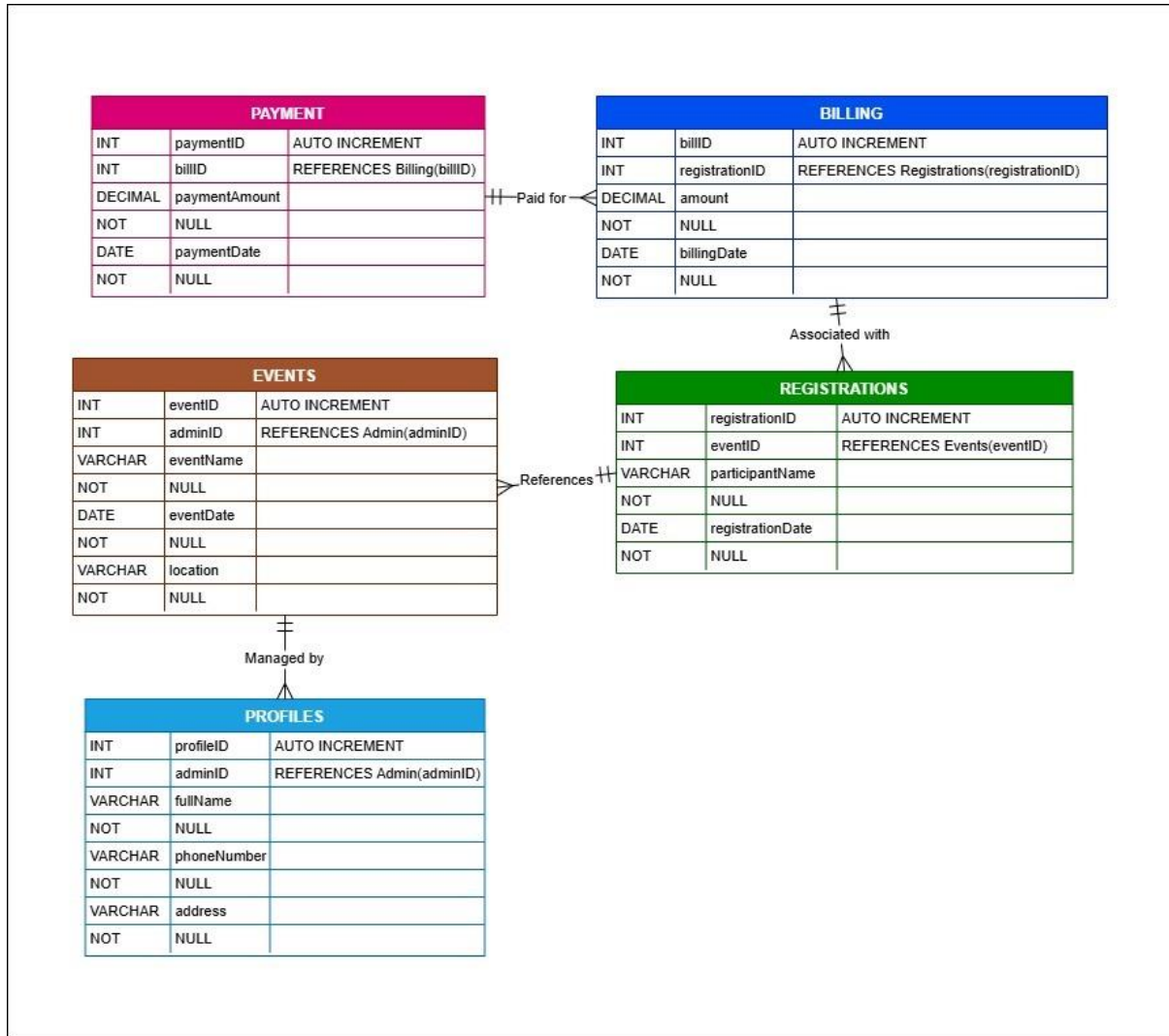
3.2 Hardware and Software Requirements

- **Hardware Requirements:**
 - Processor: Intel i3 or above.
 - RAM: 4 GB or higher.
 - Storage: Minimum 500 MB.
- **Software Requirements:**
 - Operating System: Windows/Linux/MacOS.
 - Java Development Kit (JDK).
 - MySQL Server.
 - IDE: Eclipse/IntelliJ IDEA/NetBeans.

3.3 Architecture Diagram:



3.4 ER Diagram:



3.5 Normalization

Table 1: Event Information (Before Normalization)

Event Name	Event Date	Event Location	College Name	College Location
Hackathon	2024-12-01	Auditorium	ABC College	City A
Tech Talk	2024-12-05	Conference Hall	XYZ University	City B

After Normalization - 1NF

- Each column contains atomic values.
- Separate repeating groups into individual rows.

Event Table:

eventID	eventName	eventDate	eventLocation	collegelD (FK)
1	Hackathon	2024-12-01	Auditorium	101
2	Tech Talk	2024-12-05	Conference Hall	102

College Table:

collegelD	collegeName	collegeLocation
101	ABC College	City A
102	XYZ University	City B

After Normalization - 2NF

- Remove partial dependencies by separating dependent attributes into related tables.

Event Table:

eventID	eventName	eventDate	eventLocation
1	Hackathon	2024-12-01	Auditorium
2	Tech Talk	2024-12-05	Conference Hall

College_Event Table:

eventID (FK)	collegelD (FK)
1	101
2	102

College Table:

collegelD	collegeName	collegeLocation
101	ABC College	City A
102	XYZ University	City B

After Normalization - 3NF

- Remove transitive dependencies.

User Table:

userID	userName	email	phoneNumber
201	John Doe	john@example.com	1234567890

Event Table:

eventID	eventName	eventDate	eventLocation
1	Hackathon	2024-12-01	Auditorium

Registration Table:

registrationID	userID (FK)	eventID (FK)	registrationDate
301	201	1	2024-11-15

Billing Table:

billID	userID (FK)	amount	billDate
401	201	500	2024-11-16

Payment Table:

paymentID	userID (FK)	paymentAmount	paymentDate
501	201	500	2024-11-17

College Table:

collegeID	collegeName	collegeLocation
101	ABC College	City A

4. Design and Implementation

4.1 Database Design

Entity-Relationship (ER) Model:

- **Entities:**
 - Admin, Events, Registrations, Billing, Payments, Colleges, Profiles.
- **Relationships:**
 - One-to-Many between Admin and Events.
 - One-to-Many between Events and Registrations.
 - One-to-Many between Registrations and Billing.
 - One-to-Many between Billing and Payments.
 - One-to-Many between Admin and Colleges.
 - One-to-One between Admin and Profiles.

Schema (Tables):

ADMIN TABLE:

Admin Table		
Column Name	Data Type	Constraints
adminID	INT	PRIMARY KEY, AUTO_INCREMENT
adminName	VARCHAR(255)	NOT NULL
email	VARCHAR(255)	UNIQUE, NOT NULL
password	VARCHAR(255)	NOT NULL

Events Table:

Events Table		
Column Name	Data Type	Constraints
eventID	INT	PRIMARY KEY, AUTO_INCREMENT
adminID	INT	FOREIGN KEY REFERENCES Admin(adminID)
eventName	VARCHAR(255)	NOT NULL
eventDate	DATE	NOT NULL
location	VARCHAR(255)	NOT NULL

Registrations Table:

Registrations Table

Column Name	Data Type	Constraints
registrationID	INT	PRIMARY KEY, AUTO_INCREMENT
eventID	INT	FOREIGN KEY REFERENCES Events(eventID)
participantName	VARCHAR(255)	NOT NULL
registrationDate	DATE	NOT NULL

Billing Table:

Billing Table

Column Name	Data Type	Constraints
billID	INT	PRIMARY KEY, AUTO_INCREMENT
registrationID	INT	FOREIGN KEY REFERENCES Registrations(registrationID)
amount	DECIMAL(10,2)	NOT NULL
billingDate	DATE	NOT NULL

Payments TABLE:

Payments Table

Column Name	Data Type	Constraints
paymentID	INT	PRIMARY KEY, AUTO_INCREMENT
billID	INT	FOREIGN KEY REFERENCES Billing(billID)
paymentAmount	DECIMAL(10,2)	NOT NULL
paymentDate	DATE	NOT NULL

Colleges Table:

Colleges Table

Column Name	Data Type	Constraints
collegeID	INT	PRIMARY KEY, AUTO_INCREMENT
adminID	INT	FOREIGN KEY REFERENCES Admin(adminID)
collegeName	VARCHAR(255)	NOT NULL
location	VARCHAR(255)	NOT NULL
establishedYear	YEAR	NOT NULL

Profile Table:

Profiles Table

Column Name	Data Type	Constraints
profileID	INT	PRIMARY KEY, AUTO_INCREMENT
adminID	INT	FOREIGN KEY REFERENCES Admin(adminID)
fullName	VARCHAR(255)	NOT NULL
phoneNumber	VARCHAR(15)	NOT NULL
address	VARCHAR(255)	NOT NULL

5.SOURCE CODE:

HOME PAGE:

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Home extends Application {

    @Override
    public void start(Stage primaryStage) {
        openMainWindow(primaryStage);
    }

    private void openMainWindow(Stage primaryStage) {
        // Buttons for the different pages
        Button eventsButton = new Button("Events");
        Button eventRegButton = new Button("Event Registration");
        Button billingButton = new Button("Billing");
        Button paymentButton = new Button("Payment");
        Button collegeButton = new Button("College");
        Button profileButton = new Button("Profile");

        // Set button actions to open respective pages
        eventsButton.setOnAction(e -> openEventsPage());
        eventRegButton.setOnAction(e -> openEventRegistrationPage());
        billingButton.setOnAction(e -> openBillingPage());
```



```

    paymentButton.setOnAction(e -> openPaymentPage());
    collegeButton.setOnAction(e -> openCollegePage(primaryStage));
    profileButton.setOnAction(e -> openProfilePage());

    // Layout for buttons
    VBox layout = new VBox(20);
    layout.getChildren().addAll(eventsButton, eventRegButton,
    billingButton, paymentButton, collegeButton,
    profileButton);
    layout.setAlignment(Pos.CENTER);

    // Scene setup
    Scene scene = new Scene(layout, 600, 400);
    primaryStage.setTitle("Main Menu");
    primaryStage.setScene(scene);
    primaryStage.show();
}

private Object openPaymentPage() {
    throw new UnsupportedOperationException("Unimplemented method
'openPaymentPage'");
}

// Open the Events Page
private void openEventsPage() {
    EventsPage eventsPage = new EventsPage(); // Create an instance of
EventsPage
    Scene eventsScene = new Scene(eventsPage.createPage(), 600, 400); //
Create and set the whole page scene
    Stage eventsStage = new Stage(); // New Stage for Events
    eventsStage.setTitle("Events");
    eventsStage.setScene(eventsScene);
    eventsStage.show();
}

// Open the Event Registration Page
private void openEventRegistrationPage() {
    EventRegistrationPage eventRegPage = new EventRegistrationPage();
    // Create an instance of EventRegistrationPage

```

```

    Scene eventRegScene = new Scene(eventRegPage.createPage(), 600,
400); // Create and set the whole page scene
    Stage eventRegStage = new Stage(); // New Stage for Event
Registration
    eventRegStage.setTitle("Event Registration");
    eventRegStage.setScene(eventRegScene);
    eventRegStage.show();
}

// Open the Billing Page
private void openBillingPage() {
    BillingPage billingPage = new BillingPage(); // Create an instance of
BillingPage
    Scene billingScene = new Scene(billingPage.createPage(), 600, 400); //
Create and set the whole page scene
    Stage billingStage = new Stage(); // New Stage for Billing
    billingStage.setTitle("Billing");
    billingStage.setScene(billingScene);
    billingStage.show();
}

// Open the College Page
private void openCollegePage(Stage primaryStage) {
    CollegePage collegePage = new CollegePage(); // Create an instance of
CollegePage
    Scene collegePageScene = new Scene(collegePage.createPage(), 600,
400); // Set up the scene for CollegePage
    primaryStage.setScene(collegePageScene); // Switch to the CollegePage
scene
}

// Open the Profile Page
private void openProfilePage() {
    ProfilePage profilePage = new ProfilePage(); // Create an instance of
ProfilePage
    Scene profileScene = new Scene(profilePage.createPage(), 600, 400); //
Create and set the whole page scene
    Stage profileStage = new Stage(); // New Stage for Profile
    profileStage.setTitle("Profile");
    profileStage.setScene(profileScene);
}

```

```

        profileStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

BILLING PAGE:

```

import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class BillingPage extends Application {
    private TextField userIDField, amountField;
    private DatePicker billDatePicker;
    private Button submitButton;

    @Override
    public void start(Stage primaryStage) {
        // Initialize fields and set up the layout
        userIDField = new TextField();
        userIDField.setPromptText("Enter User ID");

        amountField = new TextField();
        amountField.setPromptText("Enter Amount");

        billDatePicker = new DatePicker();

        submitButton = new Button("Submit Billing");
        submitButton.setOnAction(e -> handleBilling());

        // VBox layout for the form
    }
}

```

```

    VBox layout = new VBox(10, userIDField, amountField,
billDatePicker, submitButton);
    Scene scene = new Scene(layout, 400, 250);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Billing");
    primaryStage.show();
}

// Handle the billing process when the submit button is clicked
private void handleBilling() {
    String userID = userIDField.getText();
    String amount = amountField.getText();
    String billDate = billDatePicker.getValue().toString();

    // Perform validation
    if (userID.isEmpty() || amount.isEmpty() || billDate.isEmpty()) {
        showError("All fields must be filled out.");
    } else {
        // Insert the billing information into the database
        insertBillingIntoDatabase(userID, amount, billDate);
        showSuccess("Billing information submitted successfully.");
    }
}

// Insert billing details into the database
private void insertBillingIntoDatabase(String userID, String amount,
String billDate) {
    String url = "jdbc:mysql://localhost:3306/event_management"; //
Database URL
    String user = "root"; // MySQL username
    String password = "kabilesh"; // MySQL password

    String query = "INSERT INTO Billing (userID, amount, billDate)
VALUES (?, ?, ?)";

    try (Connection conn = DriverManager.getConnection(url, user,
password);
        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, userID);

```

```

        stmt.setString(2, amount);
        stmt.setString(3, billDate);

        stmt.executeUpdate(); // Execute the insert query

    } catch (Exception e) {
        e.printStackTrace();
        showError("Database error: Unable to save billing information.");
    }
}

// Show an error message in a dialog
private void showError(String message) {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Error");
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

// Show a success message in a dialog
private void showSuccess(String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Success");
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

// This method will return the layout as a Parent
public Parent createPage() {
    userIDField = new TextField();
    userIDField.setPromptText("Enter User ID");

    amountField = new TextField();
    amountField.setPromptText("Enter Amount");

    billDatePicker = new DatePicker();

    submitButton = new Button("Submit Billing");

```

```

        submitButton.setOnAction(e -> handleBilling());

        // Create and return the layout
        VBox layout = new VBox(10, userIDField, amountField,
billDatePicker, submitButton);
        return layout;
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

PROFILE PAGE:

```

import javafx.scene.Parent;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.VBox;
import javafx.scene.control.cell.PropertyValueFactory;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class ProfilePage {

    // TableView to display user profile data
    private TableView<UserProfile> userProfileTable;

    public ProfilePage() {
        userProfileTable = new TableView<>();
        setupUserProfileTable();
        loadUserProfilesFromDatabase();
    }

    // Method to setup the table columns
    @SuppressWarnings("unchecked")
    private void setupUserProfileTable() {

```

```
    TableColumn<UserProfile, String> userNameColumn = new  
    TableColumn<>("User Name");  
    userNameColumn.setCellValueFactory(new  
    PropertyValueFactory<>("userName"));
```

```
    TableColumn<UserProfile, String> userEmailColumn = new  
    TableColumn<>("Email");  
    userEmailColumn.setCellValueFactory(new  
    PropertyValueFactory<>("email"));
```

```
    TableColumn<UserProfile, String> userPhoneColumn = new  
    TableColumn<>("Phone");  
    userPhoneColumn.setCellValueFactory(new  
    PropertyValueFactory<>("phone"));
```

```
    userProfileTable.getColumns().addAll(userNameColumn,  
    userEmailColumn, userPhoneColumn);  
}
```

// Method to load user profiles from the database

```
private void loadUserProfilesFromDatabase() {  
    String url = "jdbc:mysql://localhost:3306/event_management";  
    String user = "root"; // Replace with your MySQL username  
    String password = "kabilesh"; // Updated password
```

```
    String query = "SELECT * FROM Users"; // Make sure your table  
    name matches the database
```

```
    try (Connection conn = DriverManager.getConnection(url, user,  
    password);
```

```
        PreparedStatement stmt = conn.prepareStatement(query);  
        ResultSet rs = stmt.executeQuery()) {
```

```
        while (rs.next()) {  
            String userName = rs.getString("userName");  
            String email = rs.getString("email");  
            String phone = rs.getString("phone");  
            userProfileTable.getItems().add(new UserProfile(userName,  
            email, phone));  
        }
```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

// Method to create the page layout and return the Parent
public Parent createPage() {
    VBox layout = new VBox(10);
    layout.getChildren().add(userProfileTable);
    return layout;
}

// UserProfile class to represent user profile data
public static class UserProfile {
    private String userName;
    private String email;
    private String phone;

    public UserProfile(String userName, String email, String phone) {
        this.userName = userName;
        this.email = email;
        this.phone = phone;
    }

    public String getUserName() {
        return userName;
    }

    public String getEmail() {
        return email;
    }

    public String getPhone() {
        return phone;
    }
}
}

```


EVENTS PAGE:

```
import javafx.scene.Parent;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.layout.VBox;  
import javafx.scene.control.cell.PropertyValueFactory;
```

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;
```

```
public class EventsPage {
```

```
    // TableView to display event data  
    private TableView<Event> eventsTable;
```

```
    public EventsPage() {  
        eventsTable = new TableView<>();  
        setupEventsTable();  
        loadEventDataFromDatabase();  
    }
```

```
    // Method to setup the table columns  
    @SuppressWarnings("unchecked")  
    private void setupEventsTable() {  
        // Column for event name  
        TableColumn<Event, String> eventNameColumn = new  
TableColumn<>("Event Name");  
        eventNameColumn.setCellValueFactory(new  
PropertyValueFactory<>("name"));
```

```
        // Column for event date  
        TableColumn<Event, String> eventDateColumn = new  
TableColumn<>("Event Date");  
        eventDateColumn.setCellValueFactory(new  
PropertyValueFactory<>("date"));
```

```
    // Add columns to the table
```

```

        eventsTable.getColumns().addAll(eventNameColumn,
eventDateColumn);
    }

    // Method to load event data from the database
    private void loadEventDataFromDatabase() {
        String url = "jdbc:mysql://localhost:3306/event_management";
        String user = "root"; // Replace with your MySQL username
        String password = "kabilesh"; // Replace with your MySQL password

        String query = "SELECT * FROM Events"; // Query to fetch all
events

        try (Connection conn = DriverManager.getConnection(url, user,
password);
            PreparedStatement stmt = conn.prepareStatement(query);
            ResultSet rs = stmt.executeQuery()) {

            // Process each event record and add it to the table
            while (rs.next()) {
                String name = rs.getString("eventName");
                String date = rs.getString("eventDate");
                eventsTable.getItems().add(new Event(name, date));
            }
        } catch (Exception e) {
            e.printStackTrace(); // Print any database errors
        }
    }

    // Method to create the page layout and return the Parent
    public Parent createPage() {
        VBox layout = new VBox(10);
        layout.getChildren().add(eventsTable);
        return layout;
    }

    // Event class to hold event data
    public static class Event {
        private String name;
        private String date;
    }

```

```

    public Event(String name, String date) {
        this.name = name;
        this.date = date;
    }

    public String getName() {
        return name;
    }

    public String getDate() {
        return date;
    }
}

```

EVENT REG PAGE:

```

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

```

```

public class EventRegistrationPage extends Application {
    private TextField participantNameField;
    private ComboBox<String> eventComboBox;
    private DatePicker registrationDatePicker;
    private Button submitButton;

```

@Override

```

    public void start(Stage primaryStage) {
        // ComboBox for selecting events
        eventComboBox = new ComboBox<>();
        loadEventsFromDatabase(); // Load event names from the database
    }

```

```

// TextField for entering participant name
participantNameField = new TextField();
participantNameField.setPromptText("Enter Participant Name");

// DatePicker for selecting registration date
registrationDatePicker = new DatePicker();

// Submit Button to handle the registration
submitButton = new Button("Register");
submitButton.setOnAction(e -> handleEventRegistration());

// Layout for the registration form
VBox layout = new VBox(10, participantNameField, eventComboBox,
registrationDatePicker, submitButton);
Scene scene = new Scene(layout, 400, 250);
primaryStage.setScene(scene);
primaryStage.setTitle("Event Registration");
primaryStage.show();
}

// Method to load event names from the database into the ComboBox
private void loadEventsFromDatabase() {
    String url = "jdbc:mysql://localhost:3306/event_management";
    String user = "root"; // Your MySQL username
    String password = "kabilesh"; // Your MySQL password

    String query = "SELECT eventName FROM Events"; // Modify this
query to match your actual event table

    try (Connection conn = DriverManager.getConnection(url, user,
password);
        PreparedStatement stmt = conn.prepareStatement(query);
        ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            String eventName = rs.getString("eventName");
            eventComboBox.getItems().add(eventName);
        }
    }
}

```

```

    } catch (Exception e) {
        e.printStackTrace();
        showError("Error loading events from database.");
    }
}

// Handle the event registration form submission
private void handleEventRegistration() {
    String participantName = participantNameField.getText();
    String selectedEvent = eventComboBox.getValue();
    String registrationDate = registrationDatePicker.getValue().toString();

    // Check if all fields are filled
    if (participantName.isEmpty() || selectedEvent == null ||
registrationDate.isEmpty()) {
        showError("All fields must be filled out.");
    } else {
        // Insert into the database (Event Registration)
        insertEventRegistrationIntoDatabase(participantName,
selectedEvent, registrationDate);
        showSuccess("Registration successful!");
    }
}

// Insert event registration data into the database
private void insertEventRegistrationIntoDatabase(String
participantName, String selectedEvent,
    String registrationDate) {
    String url = "jdbc:mysql://localhost:3306/event_management";
    String user = "root"; // Your MySQL username
    String password = "kabilesh"; // Your MySQL password

    String query = "INSERT INTO EventRegistration (participantName,
eventName, registrationDate) VALUES (?, ?, ?)";

    try (Connection conn = DriverManager.getConnection(url, user,
password);
        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, participantName);

```

```

        stmt.setString(2, selectedEvent);
        stmt.setString(3, registrationDate);

        stmt.executeUpdate(); // Execute the insert query
    } catch (Exception e) {
        e.printStackTrace();
        showError("Database error: Unable to save registration.");
    }
}

// Show an error message in a dialog
private void showError(String message) {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Error");
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

// Show a success message in a dialog
private void showSuccess(String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Success");
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

// This method returns the page layout as a Parent
public VBox createPage() {
    // ComboBox for selecting events
    eventComboBox = new ComboBox<>();
    loadEventsFromDatabase(); // Load event names from the database

    // TextField for entering participant name
    participantNameField = new TextField();
    participantNameField.setPromptText("Enter Participant Name");

    // DatePicker for selecting registration date
    registrationDatePicker = new DatePicker();

```

```

    // Submit Button to handle the registration
    submitButton = new Button("Register");
    submitButton.setOnAction(e -> handleEventRegistration());

    // Layout for the registration form
    VBox layout = new VBox(10, participantNameField, eventComboBox,
registrationDatePicker, submitButton);
    return layout;
}

public static void main(String[] args) {
    launch(args);
}
}

```

COLLEGE PAGE:

```

import javafx.scene.Parent;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.VBox;
import javafx.scene.control.cell.PropertyValueFactory;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class CollegePage {

    private TableView<College> collegeTable;

    public CollegePage() {
        collegeTable = new TableView<>();
        setupCollegeTable();
        loadCollegeDataFromDatabase();
    }

    // Method to setup the table columns

```

```

@SuppressWarnings("unchecked")
private void setupCollegeTable() {
    TableColumn<College, String> collegeNameColumn = new
TableColumn<>("College Name");
    collegeNameColumn.setCellValueFactory(new
PropertyValueFactory<>("collegeName"));

    TableColumn<College, String> collegeLocationColumn = new
TableColumn<>("Location");
    collegeLocationColumn.setCellValueFactory(new
PropertyValueFactory<>("location"));

    TableColumn<College, String> establishedYearColumn = new
TableColumn<>("Established Year");
    establishedYearColumn.setCellValueFactory(new
PropertyValueFactory<>("establishedYear"));

    collegeTable.getColumns().addAll(collegeNameColumn,
collegeLocationColumn, establishedYearColumn);
}

// Method to load college data from the database
private void loadCollegeDataFromDatabase() {
    String url = "jdbc:mysql://localhost:3306/event_management";
    String user = "root"; // Replace with your MySQL username
    String password = "kabilesh"; // Updated password

    String query = "SELECT * FROM College";

    try (Connection conn = DriverManager.getConnection(url, user,
password);
        PreparedStatement stmt = conn.prepareStatement(query);
        ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            String collegeName = rs.getString("collegeName");
            String location = rs.getString("location");
            String establishedYear = rs.getString("establishedYear");
            collegeTable.getItems().add(new College(collegeName, location,
establishedYear));
        }
    }
}

```



```

    }
} catch (Exception e) {
    e.printStackTrace(); // Log the error for debugging purposes
    showError("Error loading college data from database.");
}
}

// Method to create the page layout and return the Parent
public Parent createPage() {
    VBox layout = new VBox(10);
    layout.getChildren().add(collegeTable);
    return layout;
}

// College class to represent college data
public static class College {
    private String collegeName;
    private String location;
    private String establishedYear;

    public College(String collegeName, String location, String
establishedYear) {
        this.collegeName = collegeName;
        this.location = location;
        this.establishedYear = establishedYear;
    }

    public String getCollegeName() {
        return collegeName;
    }

    public String getLocation() {
        return location;
    }

    public String getEstablishedYear() {
        return establishedYear;
    }
}

```

```

// Show error message in case of any issues
private void showError(String message) {
    // You can create a custom alert here to notify the user about the issue.
    System.out.println(message); // Just a placeholder for now
}
}

```

PAYMENTS PAGE:

```

import javafx.scene.Parent;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.VBox;
import javafx.scene.control.cell.PropertyValueFactory;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.math.BigDecimal; // For payment amount

public class PaymentsPage {

    private TableView<Payment> paymentTable;

    public PaymentsPage() {
        paymentTable = new TableView<>();
        setupPaymentTable();
        loadPaymentDataFromDatabase();
    }

    // Method to setup the table columns
    @SuppressWarnings("unchecked")
    private void setupPaymentTable() {
        TableColumn<Payment, String> userIDColumn = new
TableColumn<>("User ID");
        userIDColumn.setCellValueFactory(new
PropertyValueFactory<>("userID"));

        TableColumn<Payment, BigDecimal> paymentAmountColumn = new
TableColumn<>("Payment Amount");
    }
}

```

```
        paymentAmountColumn.setCellValueFactory(new  
PropertyValueFactory<>("paymentAmount"));
```

```
        TableColumn<Payment, String> paymentDateColumn = new  
TableColumn<>("Payment Date");  
        paymentDateColumn.setCellValueFactory(new  
PropertyValueFactory<>("paymentDate"));
```

```
        paymentTable.getColumns().addAll(userIDColumn,  
paymentAmountColumn, paymentDateColumn);  
    }
```

```
// Method to load payment data from the database
```

```
private void loadPaymentDataFromDatabase() {  
    String url = "jdbc:mysql://localhost:3306/event_management";  
    String user = "root"; // Replace with your MySQL username  
    String password = "kabilesh"; // Replace with your MySQL password
```

```
    String query = "SELECT * FROM Payments"; // Make sure your  
table name matches the database
```

```
    try (Connection conn = DriverManager.getConnection(url, user,  
password);
```

```
        PreparedStatement stmt = conn.prepareStatement(query);  
        ResultSet rs = stmt.executeQuery()) {
```

```
        while (rs.next()) {  
            String userID = rs.getString("userID");  
            BigDecimal paymentAmount =  
rs.getBigDecimal("paymentAmount"); // Handling amount as BigDecimal  
            String paymentDate = rs.getString("paymentDate");  
            paymentTable.getItems().add(new Payment(userID,  
paymentAmount, paymentDate));  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
// Method to create the page layout and return the Parent
```

```

public Parent createPage() {
    VBox layout = new VBox(10);
    layout.getChildren().add(paymentTable);
    return layout;
}

// Payment class to represent payment data
public static class Payment {
    private String userID;
    private BigDecimal paymentAmount;
    private String paymentDate;

    public Payment(String userID, BigDecimal paymentAmount, String
paymentDate) {
        this.userID = userID;
        this.paymentAmount = paymentAmount;
        this.paymentDate = paymentDate;
    }

    public String getUserID() {
        return userID;
    }

    public BigDecimal getPaymentAmount() {
        return paymentAmount;
    }

    public String getPaymentDate() {
        return paymentDate;
    }
}
}

```

DB CONNECT:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

```

```

public class DatabaseConnection {

    // Database URL, Username, and Password
    private static final String URL = "jdbc:mysql://localhost:3306/";
    private static final String USER = "root"; // Replace with your MySQL
username
    private static final String PASSWORD = "kabilesh"; // Replace with
your MySQL password

    // Method to establish connection
    public static Connection getConnection() {
        Connection connection = null;
        try {
            // Load MySQL JDBC Driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Establish Connection
            connection = DriverManager.getConnection(URL, USER,
PASSWORD);
            System.out.println("Connection to database established
successfully!");
        } catch (ClassNotFoundException e) {
            System.err.println("MySQL JDBC Driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Failed to connect to the database.");
            e.printStackTrace();
        }
        return connection;
    }

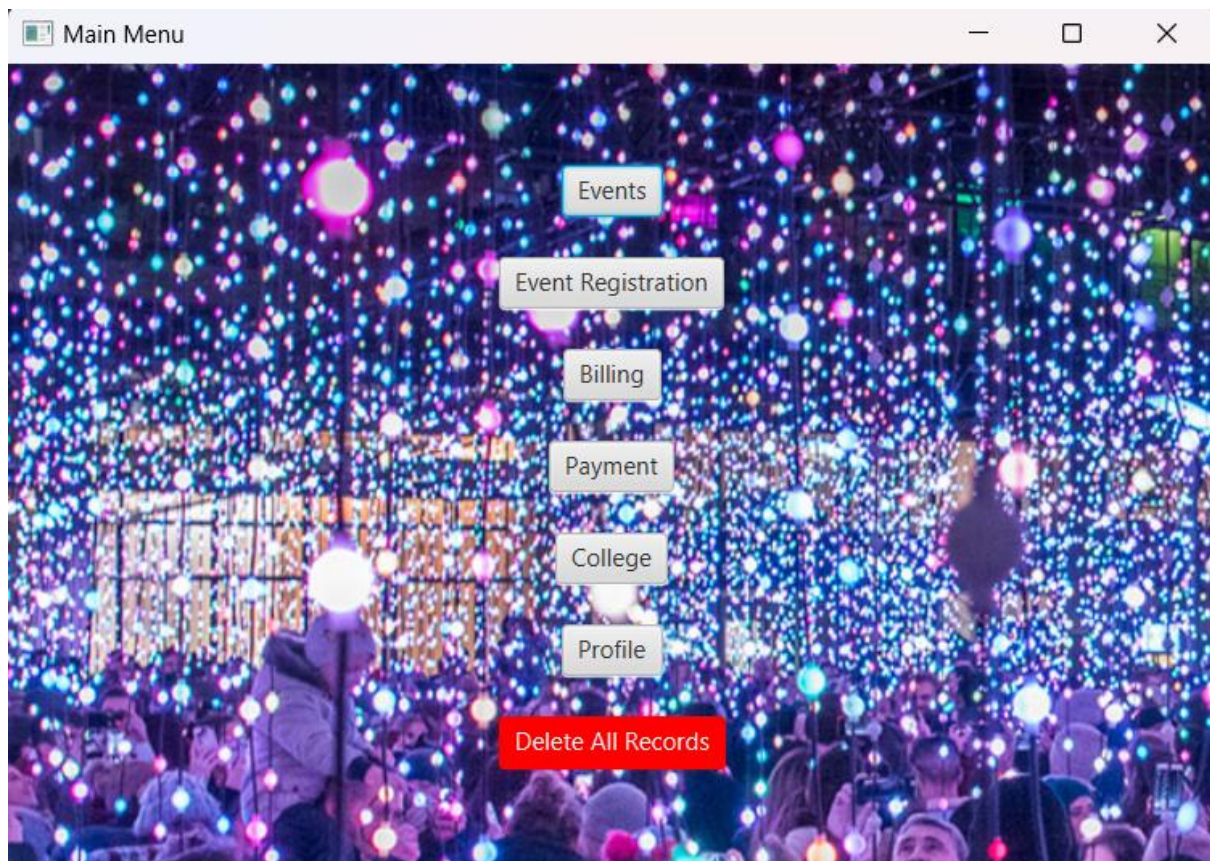
    public static void main(String[] args) {
        // Test the connection
        Connection conn = getConnection();
        if (conn != null) {
            try {
                conn.close(); // Close connection after testing
                System.out.println("Connection closed.");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```
    }  
  }  
}  
  
public static Connection connect() {  
    throw new UnsupportedOperationException("Unimplemented method  
'connect'");  
}  
}
```

6.SNAPSHOTS :

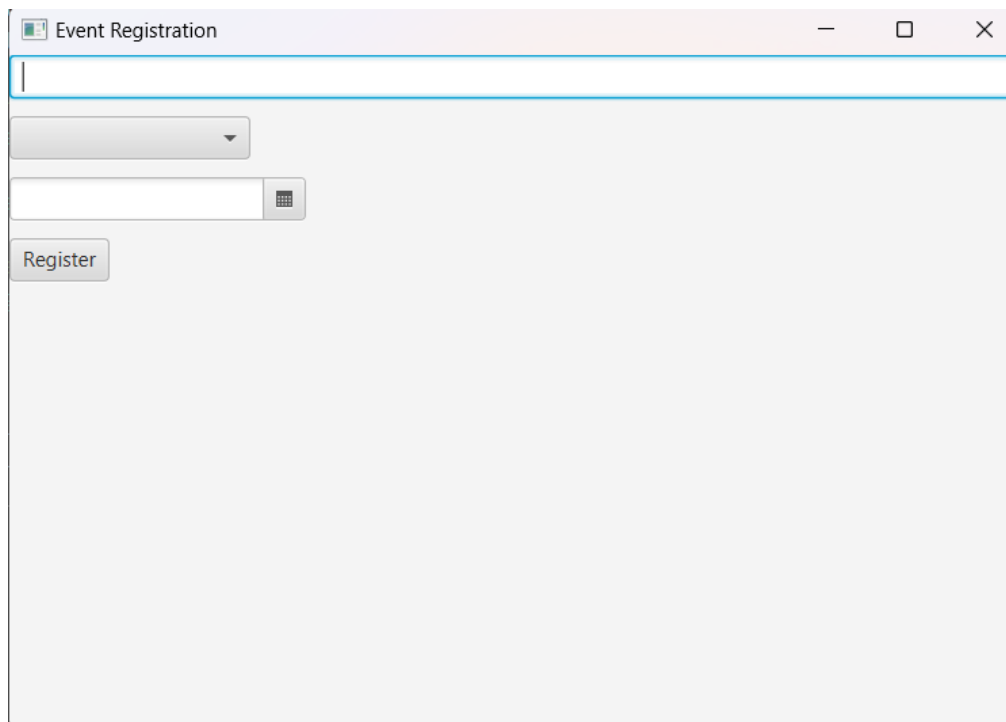
HOME PAGE:



EVENTS PAGE:

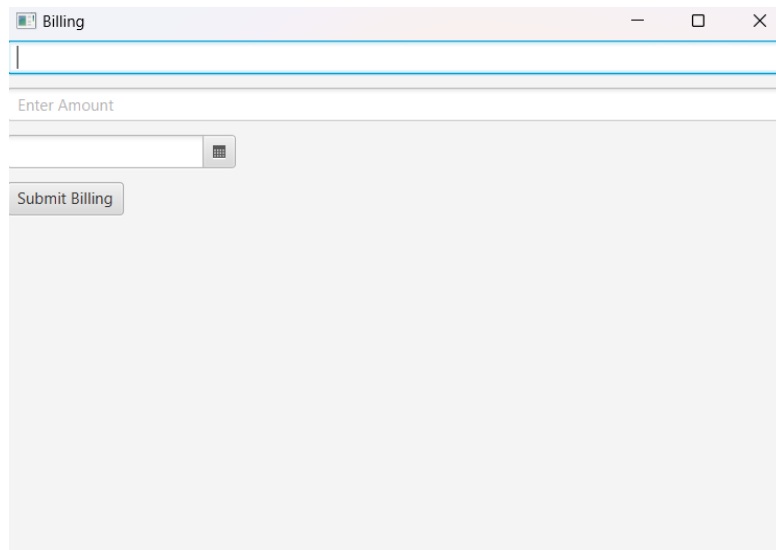
Events		
Event Name	Event Date	
Tech Talk	2024-11-25	
Hackathon	2024-12-05	
Coding Bootcamp	2025-01-15	

EVENT REG PAGE:



A screenshot of a software window titled "Event Registration". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a search bar. Underneath the search bar is a dropdown menu. Below the dropdown menu is a text input field with a calendar icon to its right. At the bottom left of the window is a "Register" button.

BILLING PAGE:



A screenshot of a software window titled "Billing". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a search bar. Underneath the search bar is a label "Enter Amount" followed by a text input field with a calendar icon to its right. At the bottom left of the window is a "Submit Billing" button.

COLLEGE PAGE:

Main Menu			
College Name	Location	Established Year	
REC	CHENNAI	1996	
SAVEETHA	CHENNAI	1990	
PANIMALAR	CHENNAI	1898	

PROFILE PAGE:

Profile			
User Name	Email	Phone	

No content in table

Conclusion:

The Event Management System (EMS) designed in this project aims to streamline the event management process, providing efficient solutions for event creation, registration, billing, payment processing, and user profile management. With the use of JavaFX for the frontend and MySQL for the database backend, the system provides a user-friendly interface for event organizers and participants. Key features such as event creation, event registration, billing, payment processing, and profile management are well-integrated into the system, allowing for seamless interaction.

The implementation of a database to store event-related information, payments, and user profiles ensures that the system can handle multiple users and events effectively. The connection between the frontend and the MySQL database through JDBC ensures data consistency and facilitates smooth communication between the application layers.

In conclusion, the EMS is a robust system designed to simplify the management of events, helping users navigate event registration, billing, and payment processes with ease. The flexibility of the design allows it to be extended with additional features such as reporting, analytics, or email notifications in the future.

References:

- 1. JavaFX Documentation**
Oracle. "JavaFX Documentation." Oracle,
<https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm>.
- 2. JDBC - Java Database Connectivity**
Oracle. "JDBC - Java Database Connectivity." Oracle,
<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>.
- 3. MySQL Documentation**
MySQL. "MySQL Reference Manual." MySQL,
<https://dev.mysql.com/doc/>.
- 4. JavaFX TableView API**
Oracle. "JavaFX TableView API." Oracle,
<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableView.html>.

Colleges Table:

Profiles Table

Column Name	Data Type	Constraints
profileID	INT	PRIMARY KEY, AUTO_INCREMENT
adminID	INT	FOREIGN KEY REFERENCES Admin(adminID)
fullName	VARCHAR(255)	NOT NULL
phoneNumber	VARCHAR(15)	NOT NULL
address	VARCHAR(255)	NOT NULL

Let me know if you'd like further customization or additional tables!

3.5 Normalization

Table 1: Event Information (Before Normalization)

After Normalization - 1NF

- Each column contains atomic values.
- Separate repeating groups into individual rows.

Event Table:

eventID	eventName	eventDate	eventLocation	collegeID (FK)
1	Hackathon	2024-12-01	Auditorium	101
2	Tech Talk	2024-12-05	Conference Hall	102

College Table:

collegeID	collegeName	collegeLocation
101	ABC College	City A
102	XYZ University	City B

After Normalization - 2NF

- Remove partial dependencies by separating dependent attributes into related tables.

Event Table:

eventID	eventName	eventDate	eventLocation
1	Hackathon	2024-12-01	Auditorium
2	Tech Talk	2024-12-05	Conference Hall

College_Event Table:

eventID (FK)	collegeID (FK)
1	101
2	102

College Table:

collegeID	collegeName	collegeLocation
101	ABC College	City A
102	XYZ University	City B

After Normalization - 3NF

- Remove transitive dependencies.

User Table:

userID	userName	email	phoneNumber
201	John Doe	<u>john@example.com</u>	1234567890

Event Table:

eventID	eventName	eventDate	eventLocation
1	Hackathon	2024-12-01	Auditorium

Registration Table:

registrationID	userID (FK)	eventID (FK)	registrationDate
301	201	1	2024-11-15

Billing Table:

billID	userID (FK)	amount	billDate
401	201	500	2024-11-16

Payment Table:

paymentID	userID (FK)	paymentAmount	paymentDate
501	201	500	2024-11-17

College Table:

collegeID	collegeName	collegeLocation
101	ABC College	City A
