

Student Management System

Abstract:

The **Student Management System** is a Java-based desktop application developed using **Java Swing** for the user interface and **MySQL** for backend data storage. The system is designed to efficiently manage student records by providing features such as adding, fetching, updating, deleting, and viewing student details.

This application replaces traditional manual record-keeping with a structured and user-friendly graphical interface. It supports all basic **CRUD operations** and enables smooth navigation between different screens. All data interactions and results are displayed directly on the GUI, ensuring ease of use, accuracy, and improved data management efficiency.

Design:

The Student Management System follows a **GUI-based client application architecture**. The system is divided into three main layers to ensure better organization, maintainability, and separation of concerns.

Presentation Layer:

Developed using Java Swing and designed with WindowBuilder, this layer provides the graphical user interface of the application. The user interfaces are created using Swing components such as **JFrame**, **JPanel**, **JButton**, **JLabel**, **JTextField**, **JTable**, and **JTextArea** through a drag-and-drop approach.

This makes the design process faster, more accurate, and easier to maintain. The presentation layer enables users to interact with the system smoothly and perform all required operations such as adding, viewing, updating, deleting, and displaying student records.

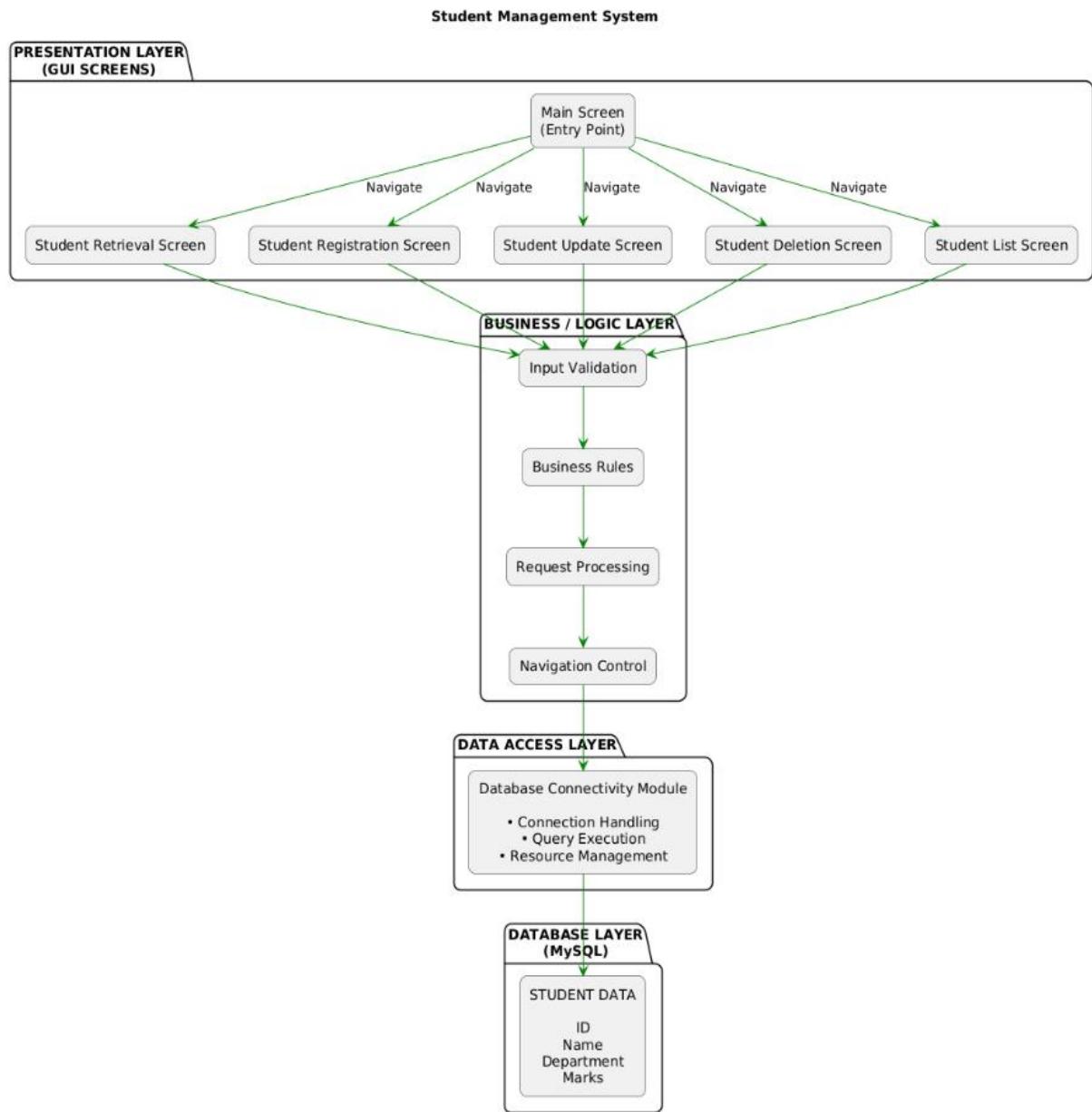
Business Logic Layer:

This layer consists of **Java classes responsible for handling application logic and database operations**. It uses **JDBC (Java Database Connectivity)** to establish communication between the application and the MySQL database.

User requests received from the presentation layer are processed, validated, and translated into appropriate **SQL queries such as INSERT, SELECT, UPDATE, and DELETE**. The results obtained from the database are then

returned to the user interface for display, ensuring accurate and reliable data handling.

Architecture

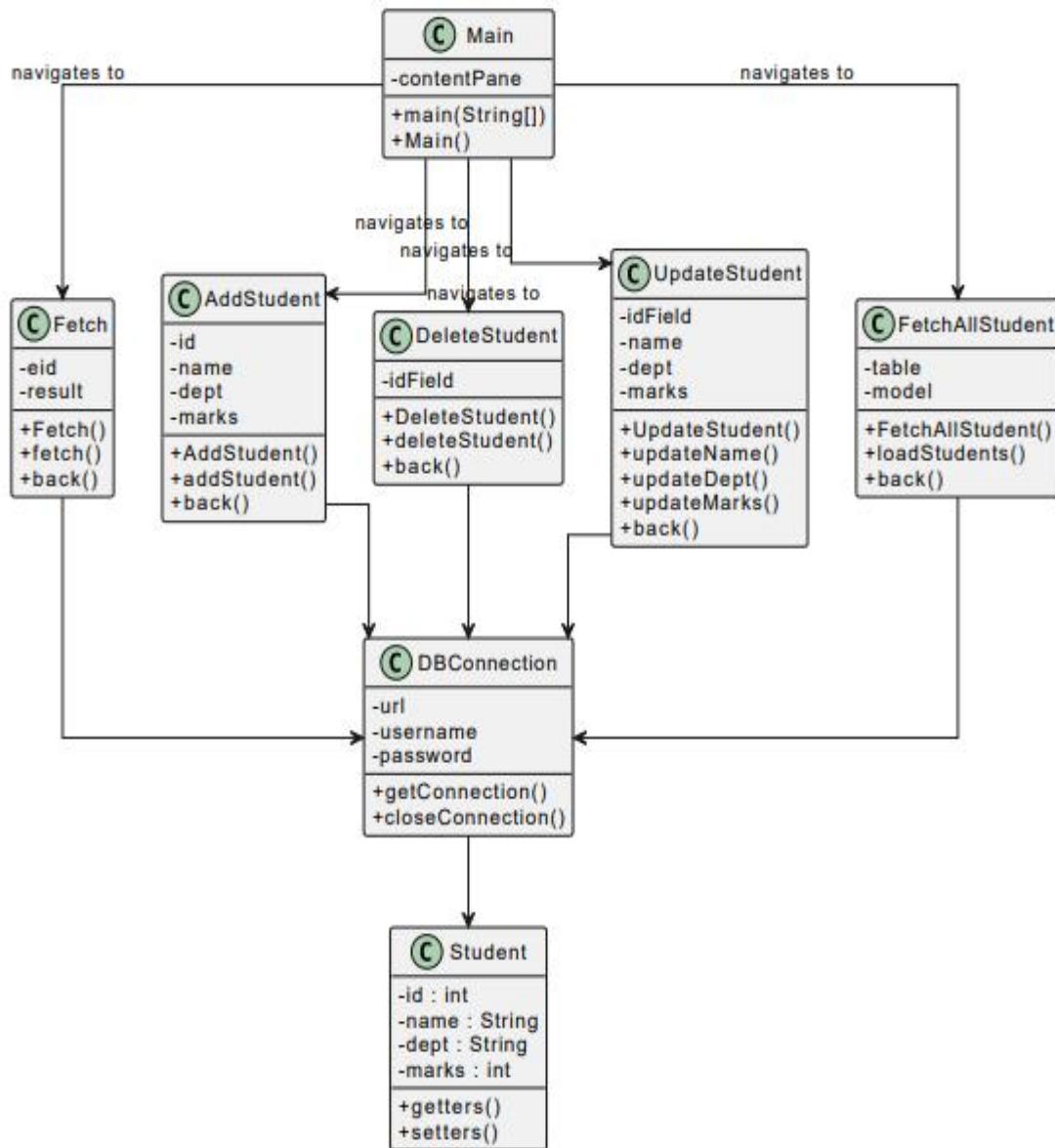


Data Layer:

The data layer uses a **MySQL database** to store and manage student records securely and efficiently. It is responsible for maintaining data persistence, ensuring that student information remains available even after the application is closed.

This layer supports fast and reliable data retrieval through structured tables and indexed fields. All data operations are performed using SQL queries, ensuring accuracy, consistency, and integrity of student records.

Class Diagram



Database Design

The system uses a **MySQL database** named **collegemanagement** with a table called **studentinfo**. The table stores essential student details in a structured format.

- id – Primary key used to uniquely identify each student,
- name – Stores the student's name,
- dept – Stores the student's department,
- marks – Stores the student's marks.

This database design ensures data integrity, avoids duplication, and supports efficient CRUD operations.

The **Main Page** acts as the central navigation hub of the application. From this page, users can access all functionalities of the Student Management System.

Components:

- **Title:** *Student Management System*
- **Buttons:**
 - Fetch Student

```
btnFetch.addActionListener(e -> {
    new Fetch();
    dispose();
});
```

- Add Student

```
btnAdd.addActionListener(e -> {
    new AddStudent();
    dispose();
});
```

- Update Student

```
btnUpdate.addActionListener(e -> {
    new UpdateStudent();
    dispose();
}) ;
```

- Delete Student

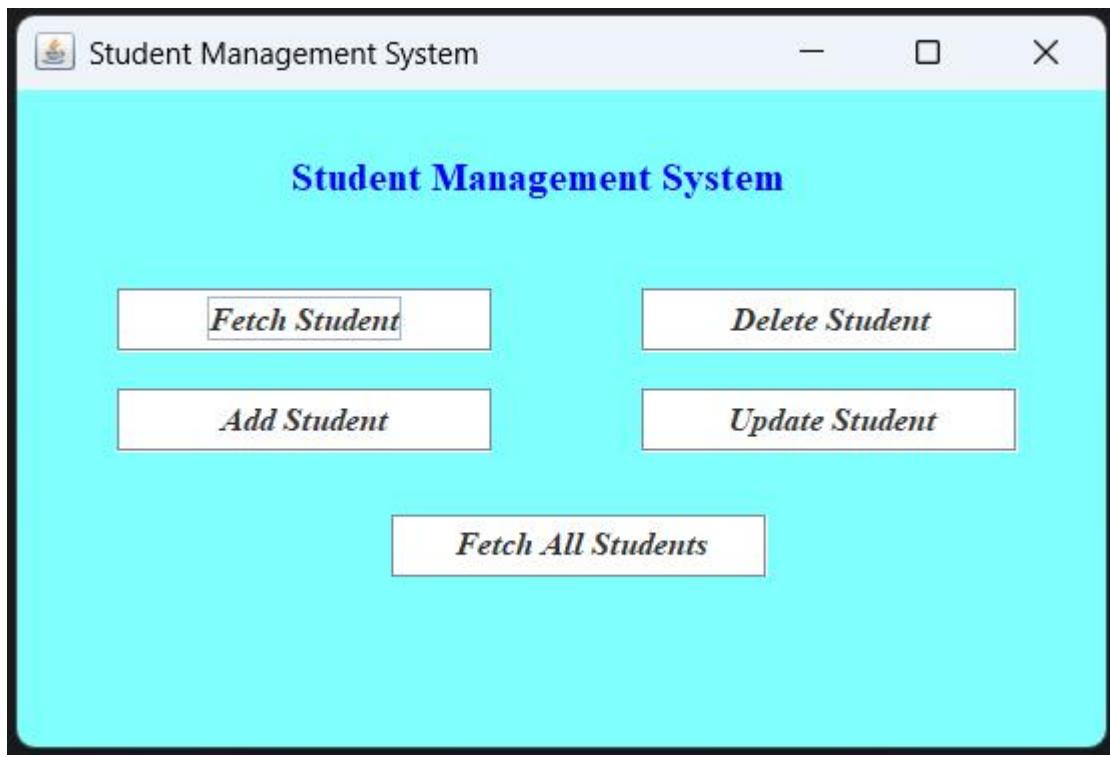
```
btnDelete.addActionListener(e -> {
    new DeleteStudent();
    dispose();
}) ;
```

- Fetch All Students

```
btnFetchAll.addActionListener(e -> {
    new FetchAllStudent();
    dispose();
}) ;
```

Functionality:

- Each button opens its respective page.
- The current page closes when a new page opens.
- Ensures smooth and controlled navigation.



Fetch Student Module

Description

This module allows the user to retrieve details of a **single student** using the student ID.

Components:

- Input field for Student ID
- Fetch button
- Text area to display student details
- Back button

```

        result.setText("");
        Connection con = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        Scanner sc = new Scanner(System.in);
        String dPath = "com.mysql.cj.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306/collegemanagement";
        String user = "root";
        String pwd = "";
        String sql = "select * from studentinfo where id = ?";
        try {
            Class.forName(dPath);
            con = DriverManager.getConnection(url, user, pwd);
            ps = con.prepareStatement(sql);
            int id = Integer.parseInt(eid.getText().trim());
            ps.setInt(1, id);

            rs = ps.executeQuery();

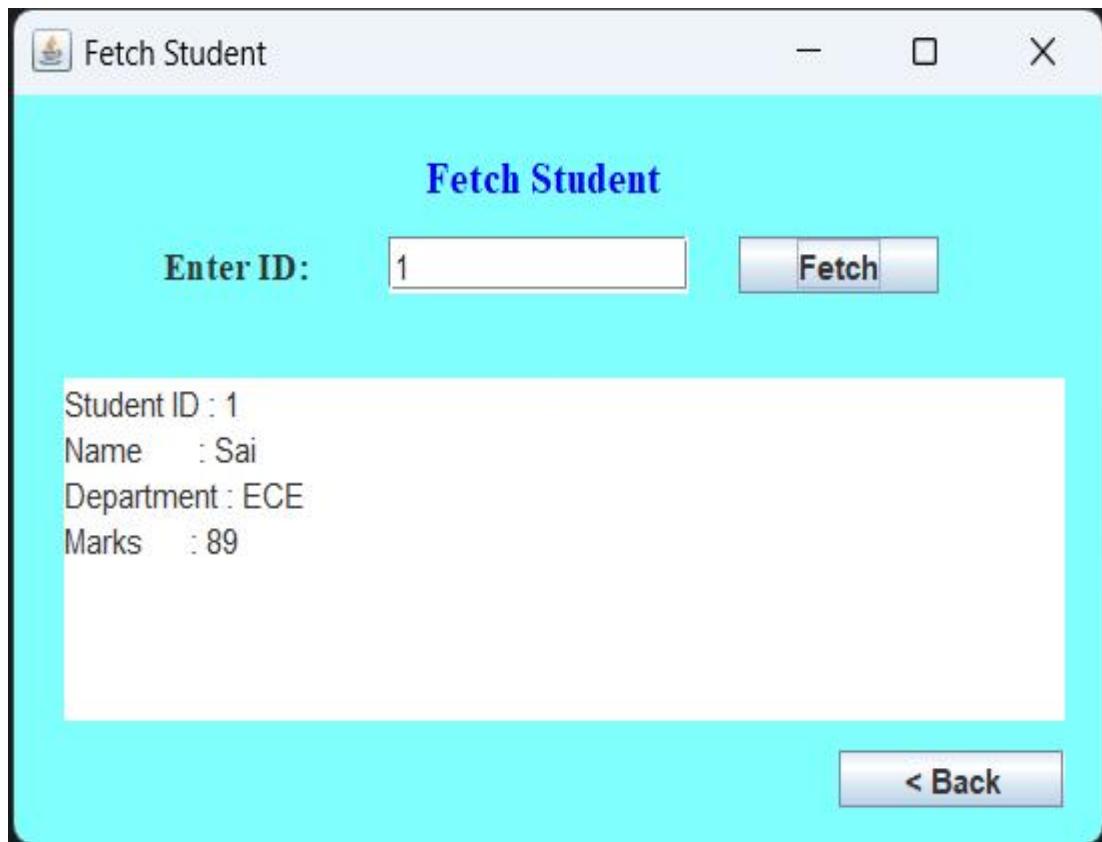
            if (rs.next()) {
                result.setText(
                    "Student ID : " + rs.getInt("id") + "\n" +
                    "Name      : " + rs.getString("name") + "\n" +
                    "Department : " + rs.getString("dept") + "\n" +
                    "Marks      : " + rs.getInt("marks")
                );
            } else {
                result.setText("Student not found");
            }
        }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if(rs != null) rs.close();
                if(ps != null) ps.close();
                if(con != null) con.close();
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

Functionality

- Accepts student ID as input
- Retrieves data from database using a prepared statement
- Displays:
 - Student ID
 - Name
 - Department
 - Marks
- Shows “Student not found” if ID does not exist
- Back button returns to Main Page



Add Student Module

Description

This module allows the user to **insert new student records** into the database.

Components

- Input fields for:
 - ID
 - Name
 - Department
 - Marks
- Add Student button
- Back button

```

Connection con = null;
PreparedStatement ps = null;
String driver = "com.mysql.cj.jdbc.Driver";
String url = "jdbc:mysql://localhost:3306/collegemanagement";
String user = "root";
String pwd = "";

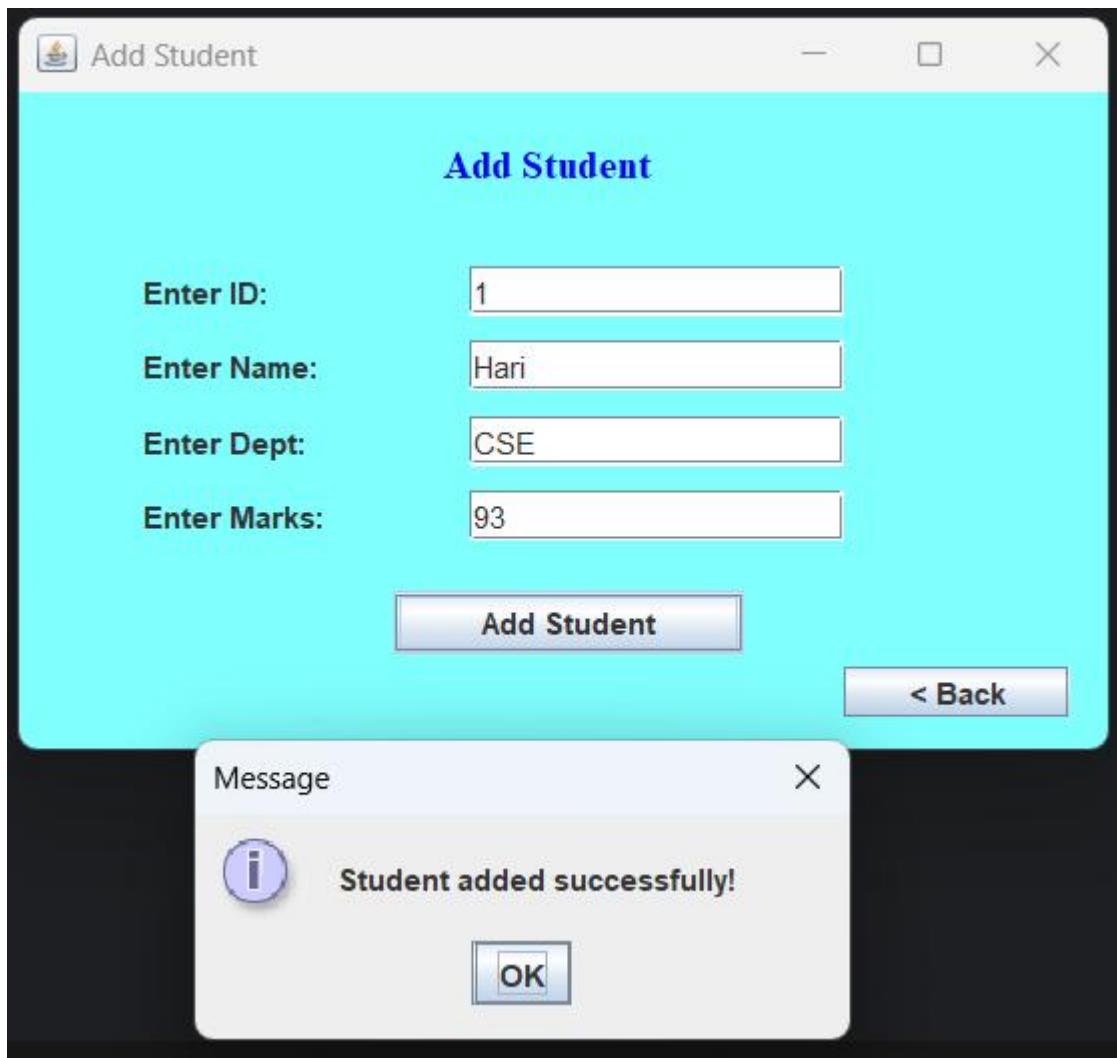
try {
    Class.forName(driver);
    con = DriverManager.getConnection(url, user, pwd);
    ps = con.prepareStatement("INSERT INTO studentinfo VALUES (?, ?, ?, ?, ?)");
    int sid = Integer.parseInt(id.getText().trim());
    int smarks = Integer.parseInt(marks.getText().trim());
    ps.setInt(1, sid);
    ps.setString(2, name.getText());
    ps.setString(3, dept.getText());
    ps.setInt(4, smarks);
    int rows = ps.executeUpdate();
    JOptionPane.showMessageDialog(
        this,
        rows > 0 ? "Student added successfully!" : "Insert failed"
    );
    // clear fields
    id.setText("");
    name.setText("");
    dept.setText("");
    marks.setText("");
} catch (Exception e1) {
    e1.printStackTrace();
} finally {
    try {
        if(con != null) con.close();
        if(ps != null) ps.close();
    } catch(Exception e1) {
        e1.printStackTrace();
    }
}
});

btnBack.addActionListener(e -> {
    new Main();
    dispose();
});
setVisible(true);

```

Functionality

- Validates that all fields are filled
- Inserts new student record into database
- Displays success or error message using dialog box
- Clears input fields after successful insertion
- Back button returns to Main Page



Update Student Module

Description

This module enables the user to **update existing student details** selectively.

Components

- Student ID input
- Input fields for:
 - New Name
 - New Department
 - New Marks
- Buttons:
 - Update Name
 - Update Department
 - Update Marks
- Back button

```

// update name
btnName.addActionListener(e -> {
    if (txtId.getText().isEmpty() || txtName.getText().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Enter ID and Name");
        return;
    }

    try (PreparedStatement ps =
            con.prepareStatement(
                "UPDATE studentinfo SET name = ? WHERE id = ?")) {

        ps.setString(1, txtName.getText());
        ps.setInt(2, Integer.parseInt(txtId.getText().trim()));

        int rows = ps.executeUpdate();
        JOptionPane.showMessageDialog(
            this,
            rows > 0 ? "Name updated successfully!" : "Student not found"
        );
    } catch (Exception ex) {
        ex.printStackTrace();
    }
});

```

```

// update department
btnDept.addActionListener(e -> {
    if (txtId.getText().isEmpty() || txtDept.getText().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Enter ID and Department");
        return;
    }

    try (PreparedStatement ps =
            con.prepareStatement(
                "UPDATE studentinfo SET dept = ? WHERE id = ?")) {

        ps.setString(1, txtDept.getText());
        ps.setInt(2, Integer.parseInt(txtId.getText().trim()));

        int rows = ps.executeUpdate();
        JOptionPane.showMessageDialog(
            this,
            rows > 0 ? "Department updated successfully!" : "Student not found"
        );
    } catch (Exception ex) {
        ex.printStackTrace();
    }
});

```

```

// update marks
btnMarks.addActionListener(e -> {
    if (txtId.getText().isEmpty() || txtMarks.getText().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Enter ID and Marks");
        return;
    }
    try (PreparedStatement ps =
            con.prepareStatement(
                "UPDATE studentinfo SET marks = ? WHERE id = ?")) {

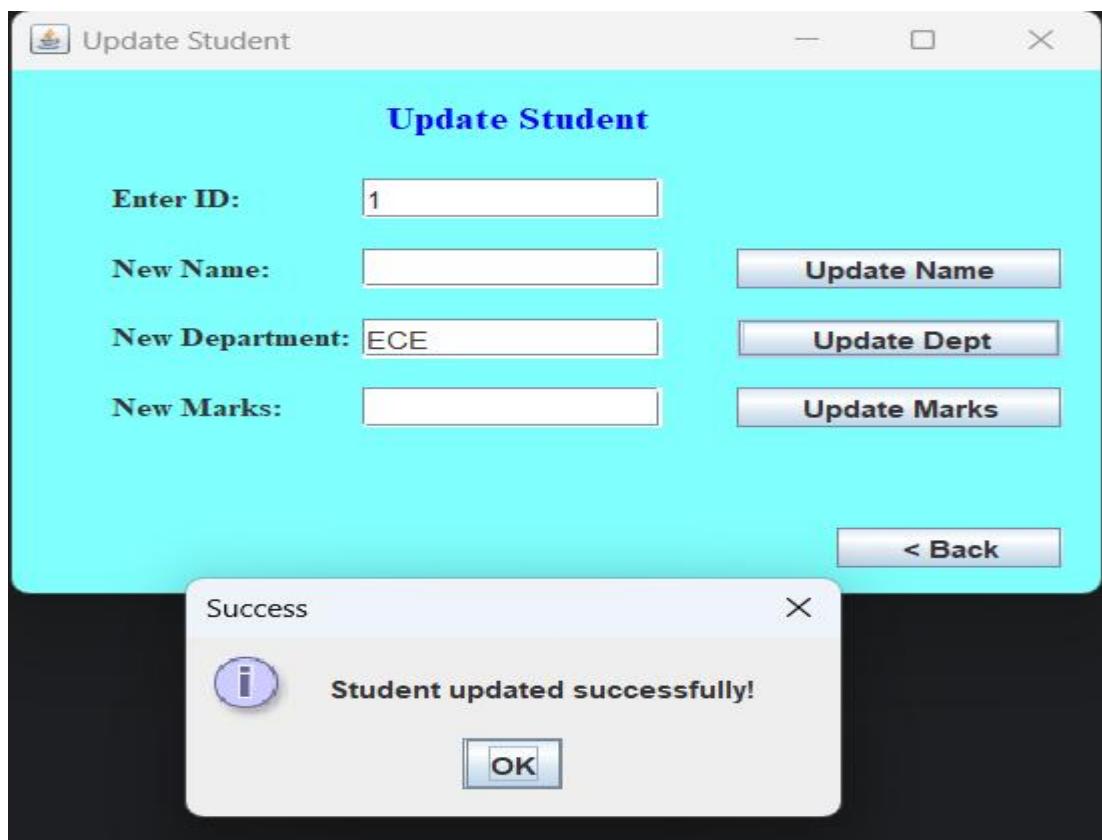
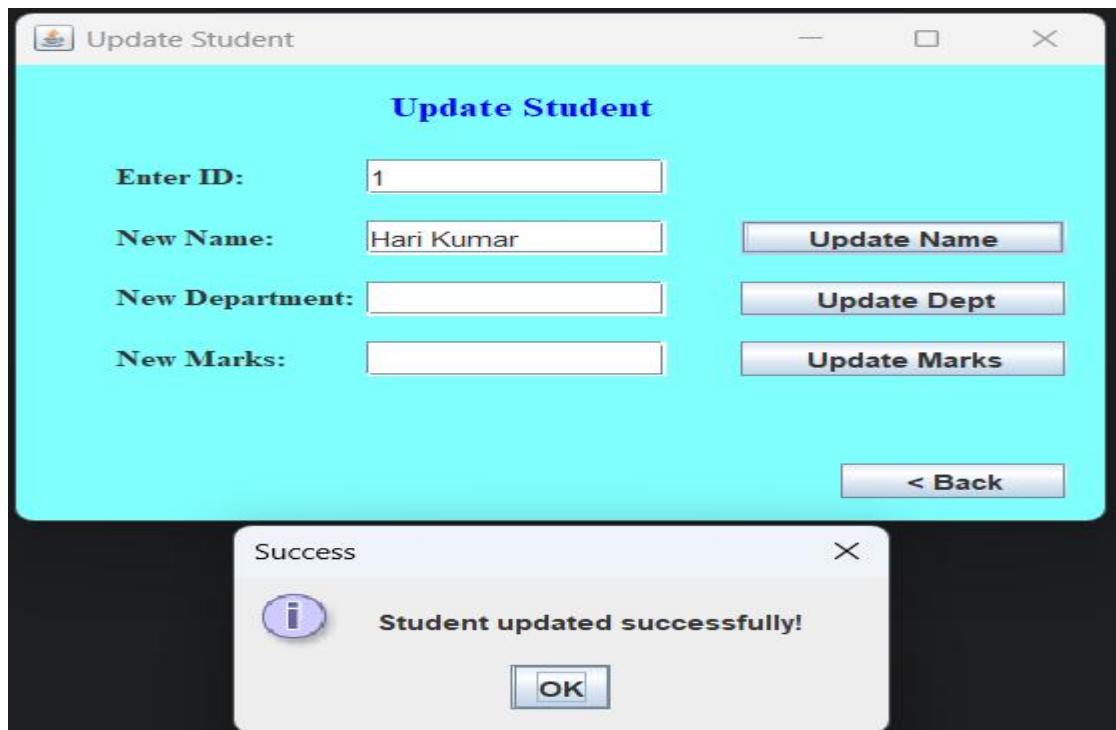
        ps.setInt(1, Integer.parseInt(txtMarks.getText().trim()));
        ps.setInt(2, Integer.parseInt(txtId.getText().trim()));

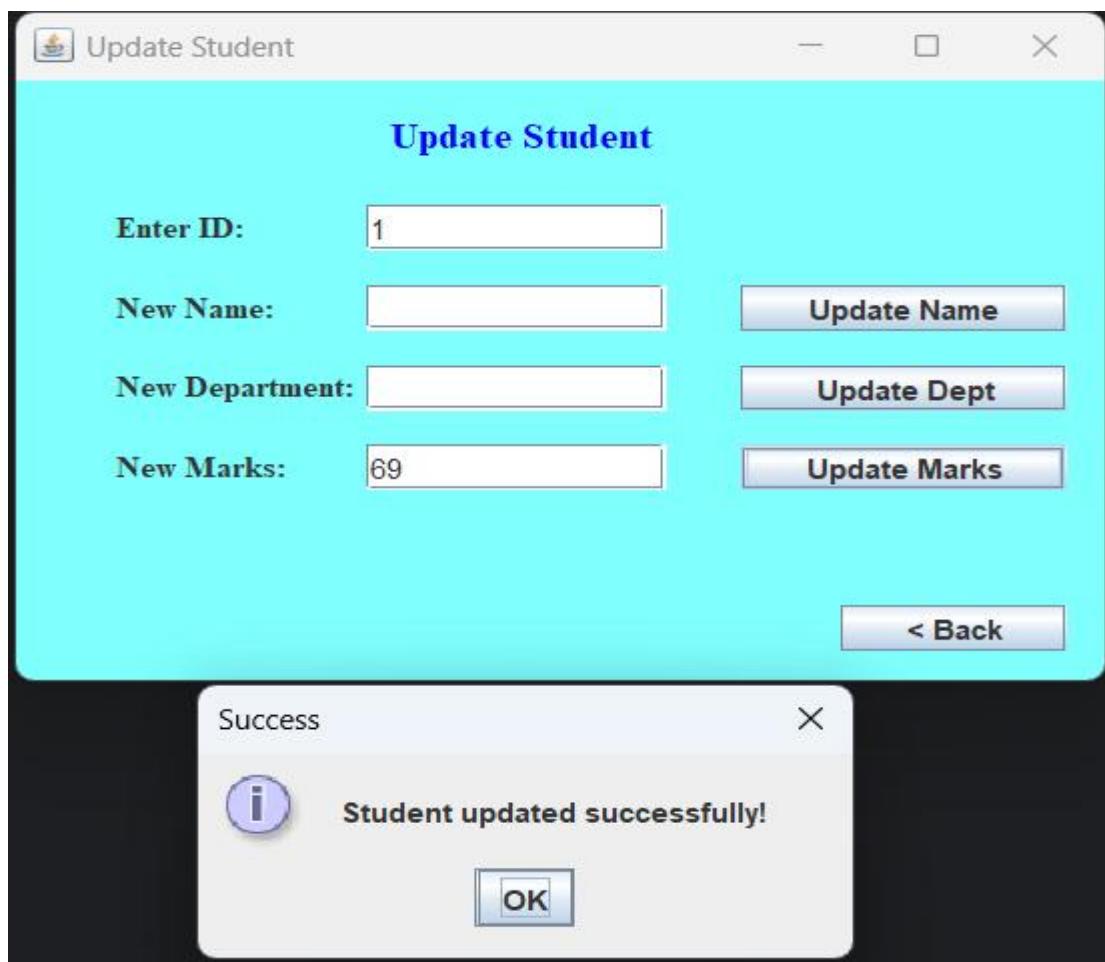
        int rows = ps.executeUpdate();
        JOptionPane.showInputDialog(
            this,
            rows > 0 ? "Marks updated successfully!" : "Student not found"
        );
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, ex.getMessage());
    }
});
btnBack.addActionListener(e -> {
    try {
        if (con != null) con.close(); // ⚡ close once
    } catch (SQLException ignored) {}
    new Main();
    dispose();
});
setVisible(true);

```

Functionality

- User can update individual attributes without affecting others
- Executes update queries using prepared statements
- Displays success or failure messages
- Back button navigates to Main Page





Delete Student Module

Description

This module allows the user to **delete a student record** using the student ID.

Components

- Student ID input field
- Delete button
- Back button

```

Connection con = null;
PreparedStatement ps = null;
try {
    int id = Integer.parseInt(txtId.getText().trim());

    con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/collegemanagement",
        "root", ""
    );
    ps = con.prepareStatement("DELETE FROM studentinfo WHERE id = ?");
    ps.setInt(1, id);
    int rows = ps.executeUpdate();
    JOptionPane.showMessageDialog(
        this,
        rows > 0 ? "Student deleted successfully!"
            : "Student not found"
    );

    txtId.setText("");
} catch (Exception ex) {
    ex.printStackTrace();
}finally {
    try {
        if(con != null) con.close();
        if(ps != null) ps.close();
    } catch(Exception e1) {
        e1.printStackTrace();
    }
}
});

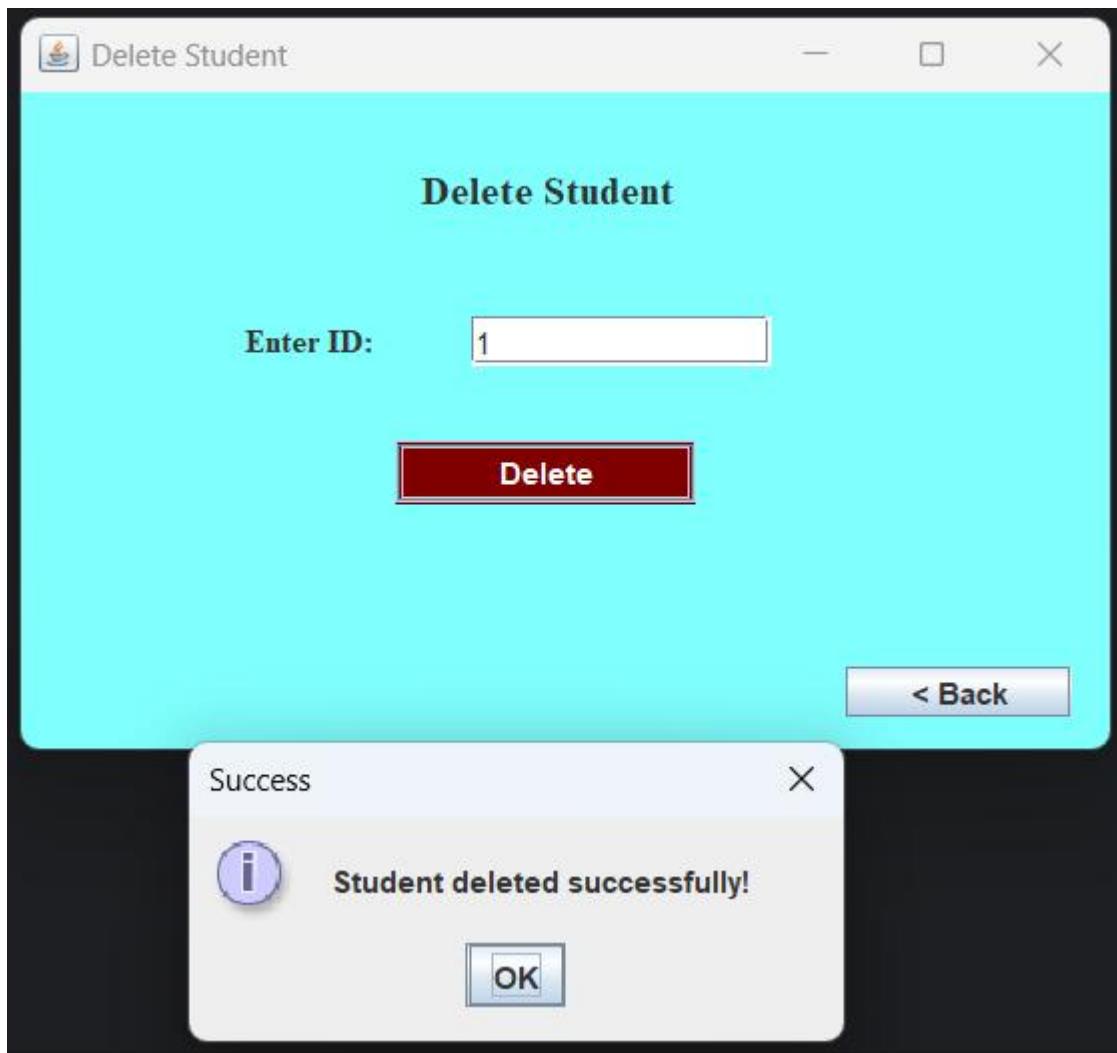
btnBack.addActionListener(e -> {
    new Main();
    dispose();
});

setVisible(true);

```

Functionalities

- Deletes record from database if ID exists
- Shows confirmation message after deletion
- Handles invalid ID cases
- Back button returns to Main Page



Fetch All Students Module

Description

This module displays **all student records** in a tabular format.

Components

- JTable with columns:
 - ID
 - Name
 - Department
 - Marks
- Scroll pane
- Back button

```

try {
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/collegemanagement",
        "root",
        ""
    );

    PreparedStatement ps =
        con.prepareStatement("SELECT * FROM studentinfo");
    ResultSet rs = ps.executeQuery();

    while (rs.next()) {
        model.addRow(new Object[]{
            rs.getInt("id"),
            rs.getString("name"),
            rs.getString("dept"),
            rs.getInt("marks")
        });
    }

    rs.close();
    ps.close();
    con.close();
}

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

```

Functionality

- Retrieves all student records from database
- Displays data in JTable
- Uses **alternate row colors** for better readability
- Supports vertical scrolling
- Back button navigates to Main Page

The screenshot shows a Windows application window titled "Fetch All Students". The main title bar is light blue with the window title. Below it, the main content area has a teal header bar with the text "FETCH ALL STUDENTS" in bold blue capital letters. The main body is a table with the following data:

ID	Name	Department	Marks
1	Hari Kumar	ECE	69
2	Ramesh	CIVIL	82
3	Vyshnavi	CSE	88
4	Amrutha	CSE	100
5	Sudheer	AIDS	70
6	Mahesh	CSE	85
7	Harsha	MECH	96
8	Ganesh	ECE	90

In the bottom right corner of the main content area, there is a small button labeled "< Back". The window has standard window controls (minimize, maximize, close) at the top right.

Future Enhancement

- **User Authentication System**
Add secure login with role-based access for admin, faculty, and students.
- **Improved User Interface**
Enhance the GUI with better layouts, validations, and modern design elements.
- **Advanced Search and Reports**
Implement search, filter, and sorting options along with report generation.
- **Data Export and Backup**
Enable exporting data to PDF/Excel and implement database backup features.
- **Attendance and Performance Tracking**
Add attendance management and student performance analysis modules.
- **Web or Cloud Migration**
Convert the desktop application into a web-based or cloud-hosted system.

Conclusion

The Student Management System demonstrates the effective use of **Java Swing** and **MySQL** to build a reliable desktop application for managing student records. It simplifies data management by replacing manual processes with a structured, database-driven approach.

Through this project, key concepts such as **GUI design, JDBC connectivity, event handling, and modular programming** were learned. The system provides a strong base for developing more advanced management applications in the future.