ABSTRACT

In the dynamic landscape of modern education, the effective management of university resources and student information is paramount. This abstract presents a comprehensive relational database management system (RDBMS) project aimed at facilitating the management of student, teacher, and department details, along with an integrated examination section and fee payment functionality. The system caters to the intricate needs of university administration by providing robust features for adding, updating, and retrieving essential information.

The core functionalities of the system include:

1. **Student Management**: The system allows administrators to seamlessly add and update student details, including personal information, academic records, and contact information. This ensures accurate and up-to-date student profiles, enabling efficient communication and record-keeping.

2. **Teacher Management** Faculty information, including qualifications, courses taught, and contact details, can be easily managed through the system. Administrators can update teacher profiles, assign courses, and maintain a comprehensive database of faculty members.

3. **Department Management**: The system facilitates the management of departmental information, including department heads, courses offered, and staff details. Administrators can efficiently organize departmental resources and streamline administrative processes.

4. **Examination Section**: A dedicated module within the system enables administrators to input and manage examination results. Students and faculty members can access their respective examination records, facilitating academic assessment and performance tracking.

5. **Fee Payment Facility**: The system provides a convenient platform for students to pay their fees securely online. Administrators can track fee payments and manage financial transactions effectively.

By leveraging the power of relational databases, the University Management System offers scalability, reliability, and data integrity. It empowers university administrators with the tools they need to streamline administrative workflows, enhance transparency, and deliver superior academic experience to students and faculty alike. With its user-friendly interface and robust functionality, the system serves as an asset in the modernization of university management processes.

TABLE OF CONTENTS

INTRODUCTION

In today's fast-paced educational environment, managing student, teacher, and department details efficiently is crucial for smooth university operations. This project, The University Management System, aims to address these challenges by providing a comprehensive system that simplifies the process of adding, updating, and managing student, teacher, and department information. Additionally, this system includes features such as an examination section to display student marks and a fee payment facility for convenient student fee management.

PROBLEM DESCRIPTION

In many universities, the traditional methods of managing student, teacher, and department information are often manual, time-consuming, and prone to errors. This can lead to inefficiencies in administrative tasks and difficulties in accessing accurate information when needed. Moreover, tracking student examination marks and managing fee payments can be tedious tasks, further adding to the administrative burden.

This project seeks to solve these problems by introducing a user-friendly RDBMS-based University Management System. By digitizing student, teacher, and department information, we aim to streamline administrative processes and improve data accuracy. The inclusion of an examination section will enable easy access to student marks, facilitating academic assessment. Additionally, the fee payment facility will provide students with a convenient way to pay their fees while allowing administrators to track payments efficiently.

Overall, this project aims to modernize university management processes, enhance transparency, and provide a better experience for students, teachers, and administrators alike.

SOFTWARE & HARDWARE REQUIREMENTS

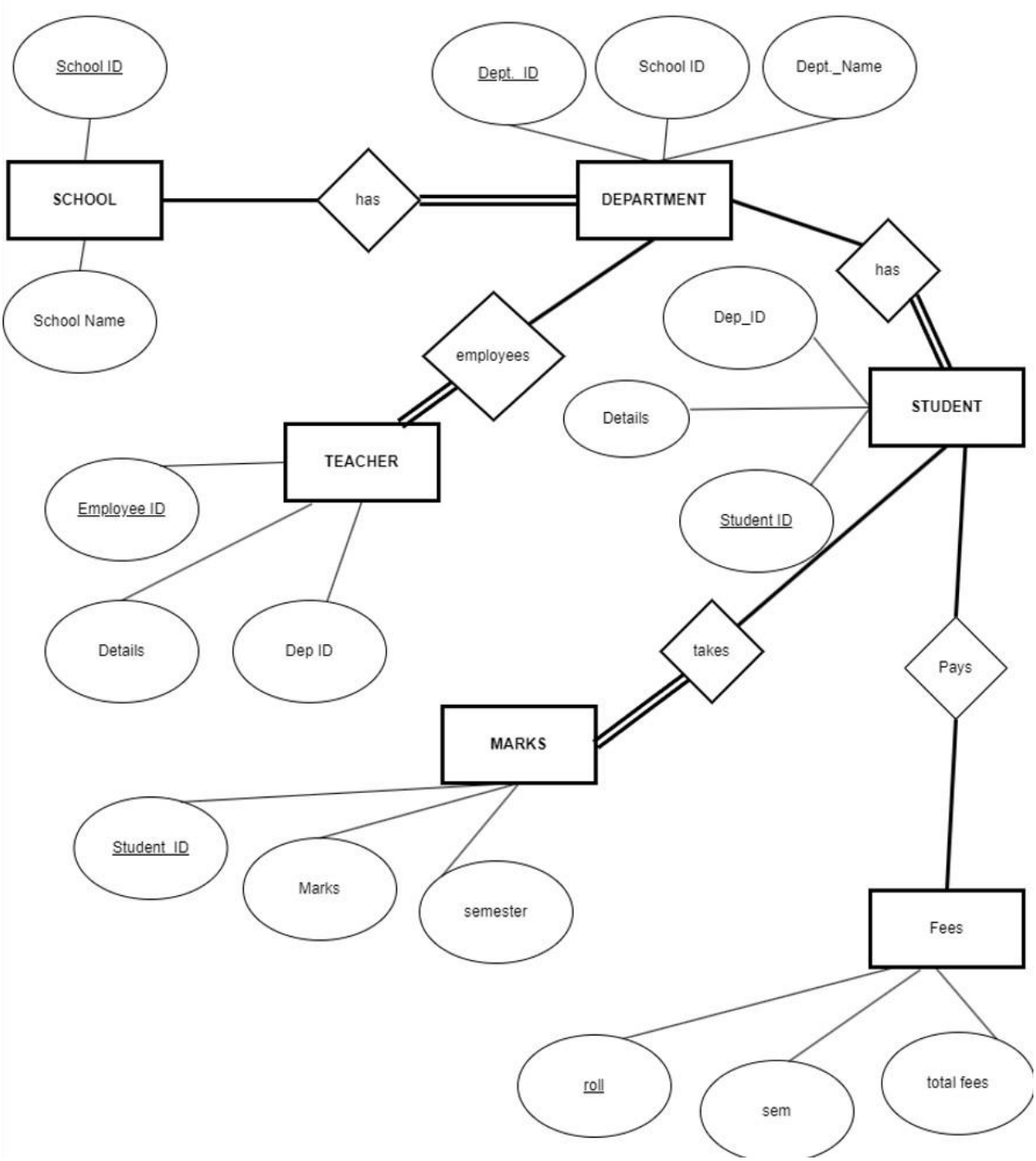
Software Requirements:

Operating System: Windows 10 or later, macOS, or Linux
Database Management System: MySQL.
Programming Languages:
Backend: Java
Frontend: Java Version Control: Git
Integrated Development Environment (IDE): IntelliJ IDEA

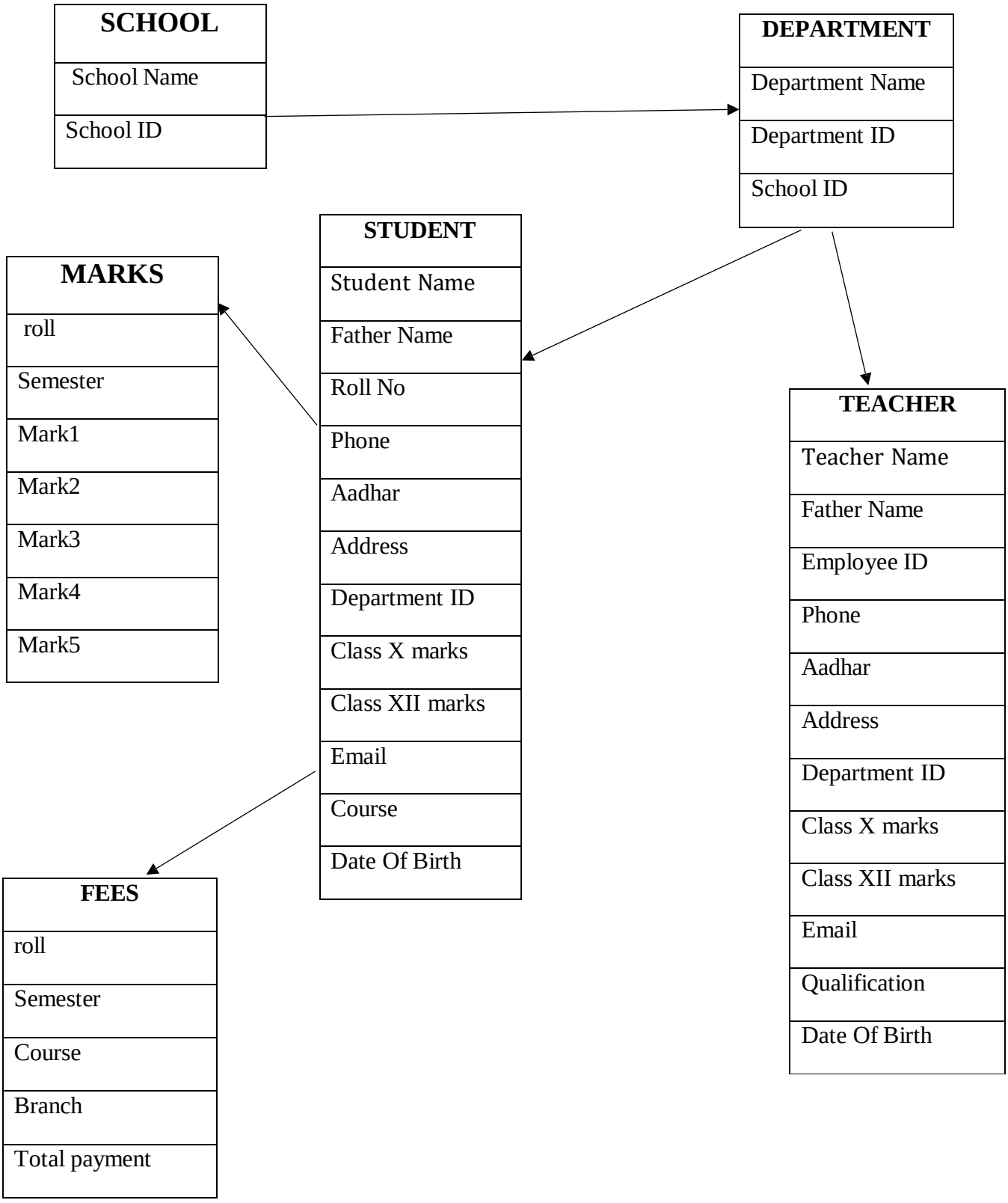
Hardware Requirements:

Processor: Intel Core i5 or equivalent
RAM: Minimum 8GB (16GB recommended for better performance)
Storage: Minimum 256GB SSD (HDD can be used but SSD is recommended for better speed)
Network: Ethernet connection for stable server hosting, Wi-Fi for client devices
Display: Minimum 15-inch monitor with a resolution of 1920x1080 pixels
Input Devices: Keyboard and Mouse for server and admin PCs.

ENTITY - RELATIONSHIP MODEL



RELATIONAL MODEL



KEY OPERATIONS IN DATABASE

CREATION OF TABLES(Below are some of the tables of the project)

```
create table school (
school_name varchar(250) not null,
school_id varchar(50) ,
primary key(school_id)
 );
```

This creates a table named school with attributes school_name and school_id

school_name
Data Type: VARCHAR(250)
Constraints: NOT NULL
Description: Stores the name of the school

school_id
Data Type: VARCHAR(50)
Constraints: PRIMARY KEY
Description: Unique identification number for each school. It serves as the primary key for this table.

Description: This SCHOOL table will allow the university to maintain a record of its school, including their names and ID, ensuring each school has a unique identification.

```
create table department(
dep_name varchar(50),
dep_id varchar(50),
school_id varchar(50),
primary key (dep_id),
foreign key (school_id) references school(school_id)
 );
```

This creates a table named department with attributes dep_name , dep_id , school_id.

Dep_id
Data Type: VARCHAR(50)
Constraints: PRIMARY KEY
Description: Unique Department. It serves as the primary key for this table.

Dep_name
Data Type: VARCHAR(50)
Description: Stores the name of department.

School_id
Data Type: VARCHAR(50)
Constraints: FOREIGN KEY references school_id of school table.
Description: It denotes the school id which was Primary key of school table, and every department is linked to its respective school.

This DEPRTMENT table will allow the users to take a note of the number and name of departments present Iin the school and hence, the university.

```
create table student (
Sname varchar(50), Fname varchar(50),
roll varchar(50), dob varchar(50),
address varchar(50), phone varchar(50),
email varchar(50), cl_x varchar(50),
cl_xii varchar(50), aadhar varchar(50),
course varchar(50), dep_id varchar(50),
primary key(roll),
foreign key (dep_id) references department(dep_id)
);
```

This creates a table named STUDENT with attributes Student name, Father Name, roll, dob, phone, class X, aadhar, dep_id, address, email, class xii, course.

```
create table teacher (
Tname varchar(50), Fname varchar(50),
empID varchar(50), dob varchar(50),
address varchar(50), phone varchar(50),
email varchar(50), cl_x varchar(50),
cl_xii varchar(50), aadhar varchar(50),
edu varchar(50), dep_id varchar(50),
primary key(empID),
foreign key (dep_id) references department(dep_id)
);
```

This creates a table named TEACHER with attributes Teacher name, Father Name, employee id, dob, phone, class X, aadhar, dep_id, address, email, class xii, course.

```
create table marks(
roll varchar(50),
sem varchar(50),
mark1 varchar(50),
mark2 varchar(50),
mark3 varchar(50),
mark4 varchar(50),
mark5 varchar(50)
);
```

This creates a table named MARKS with attributes roll, semester, mark1, mark2, mark3, mark4, mark5.

```
create table feecollege(
roll varchar(50),
course varchar(50),
branch varchar(50),
sem varchar(20),
total varchar(20)
);
```

This creates a table named FEESCOLLEGE with attributes roll, course, branch, sem, total payment.

INSERTION IN TABLE(Below are some insertion of the tables of the project)

```
insert into department values ('Computer and Communication Engineering', 'CCE'),
                              ('Computer Science Engineering', 'CSE'),
                              ('Electrical and Electronics Engineering', 'EEC'),
                              ('Mechanical Engineering', 'ME'),
                              ('Internet Of Things', 'IOT'),
                              ('Artificial Intelligence and Machine Learning', 'AIML'),
                              ('Civil Engineering', 'CE'),
                              ('Information Technology', 'IT'),
                              ('Data Science', 'DS');
```

Insert into DEPARTMENT Table:

Nine Departments have been inserted in this query.

```
insert into school values ('School of Computer and Communication Engineering', 'SCCE'),
                          ('School of Computer Science Engineering', 'SCSE'),
                          ('School of Mechanical Engineering', 'SME'),
                          ('School of Civil Engineering', 'SCE'),
                          ('School of Electronics Engineering', 'SEE'),
                          ('School of Information Technology', 'SIT');
```

Insert into SCHOOL Table:

Six schools have been inserted into the table by using insert query.

UPDATION IN TABLE(Below are some updation of the tables of the project)

```
update department set school_id = 'SCCE' where dep_id = 'CCE';
update department set school_id = 'SCSE' where dep_id = 'CSE';
update department set school_id = 'SEE' where dep_id = 'EEC';
update department set school_id = 'SME' where dep_id = 'ME';
update department set school_id = 'SCCE' where dep_id = 'IOT';
update department set school_id = 'SCSE' where dep_id = 'AIML';
update department set school_id = 'SCE' where dep_id = 'CE';
update department set school_id = 'SIT' where dep_id = 'IT';
update department set school_id = 'SIT' where dep_id = 'DS';
```

Department table Update:

The Department table is updated several times using the dep_id column.

```
alter table student add column dep_id varchar(50);
alter table student drop column branch;

alter table student add constraint primary key(roll);
```

Student table Alter:

The student table is altered several times using the dep_id column and branch column.

FRONT END

SPLASH SCREEN

The Splash Screen will be displayed first and it will be displayed till 7 seconds. It will then redirected to the Login Page.

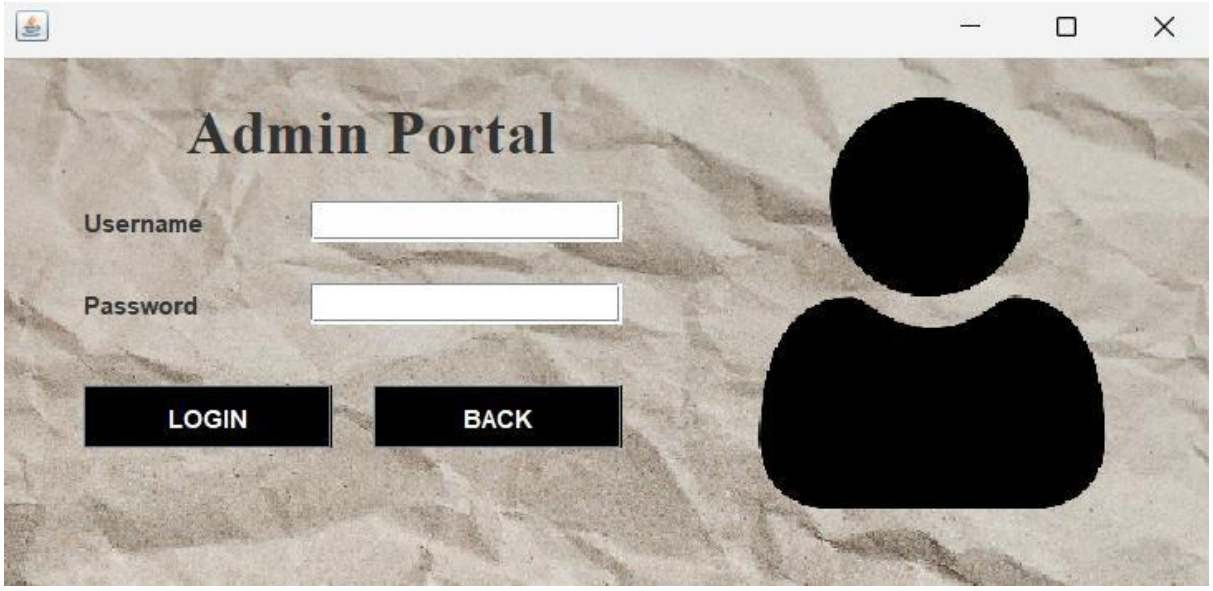
A splash screen, in the context of front-end development, serves as an introductory screen that appears when an application or website is launched. It provides users with a visual cue that the application is loading and sets the tone for their experience. Here's a breakdown of its significance and implementation:

In summary, a splash screen serves as a gateway to your application or website, offering an opportunity to showcase your brand, engage users, and provide feedback during the loading process. By carefully designing and implementing a splash screen, you can create a positive first impression and enhance the overall user experience.

SPLASH SCREEN CODE (java):

```
package university.manangement.system;

import javax.swing.*;
import java.awt.*;
public class splash extends JFrame implements Runnable{
    Thread t;
    splash(){
        JLabel heading = new JLabel("MANIPAL UNIVERSITY Admin Portal");
        heading.setBounds(50, 10, 1000, 60);
        heading.setFont(new Font("Tahoma", Font.BOLD, 50));
        heading.setForeground(Color.black);
        add(heading);

        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("icon/first.png"));
        Image i2 =
i1.getImage().getScaledInstance(1000,700,Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel img = new JLabel(i3);
        add(img);
        t = new Thread(this);
        t.start();

        setVisible(true);
        int x=1;
        for(int i = 2;i<=600;i+=4, x+=1){
            setLocation(600-((i+x)/2), 350-(i/2));
            setSize(i +3*x, i+x/2);

            try{
                Thread.sleep(10);
            }catch(Exception e){
                e.printStackTrace();
            }

        }
    }
    public void run(){
        try {
            Thread.sleep(7000);
            setVisible(false);
            new login();

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

    public static void main(String[] args) {
        new splash();
    }
}
```

LOGIN PAGE



Implementing a LOGIN page for the RDBMS project using Java and Java Frames offers a secure and user-friendly way for users to access the system. Here's a detailed overview of its significance and implementation:

Authentication: The LOGIN page serves as the primary authentication mechanism for users to access the University Management System. It verifies the identity of users by prompting them to enter their credentials, such as username and password, before granting access to the system. This helps ensure that only authorized users can interact with sensitive data within the system.

User Interface: The LOGIN page is designed using Java Frames to provide a visually appealing and intuitive user interface. Java Frames offer flexibility in designing the layout, incorporating graphical elements such as labels, text fields, buttons, and images to create a polished and professional appearance. The layout can be customized to align with the branding and design principles of the University Management System.

Input Validation: Input validation is crucial to ensure the security and integrity of the LOGIN page. Java allows for robust input validation techniques to be implemented, such as checking for the presence of required fields, validating the format of input data (e.g., ensuring passwords meet complexity requirements), and preventing common security vulnerabilities, such as SQL injection or cross-site scripting (XSS).

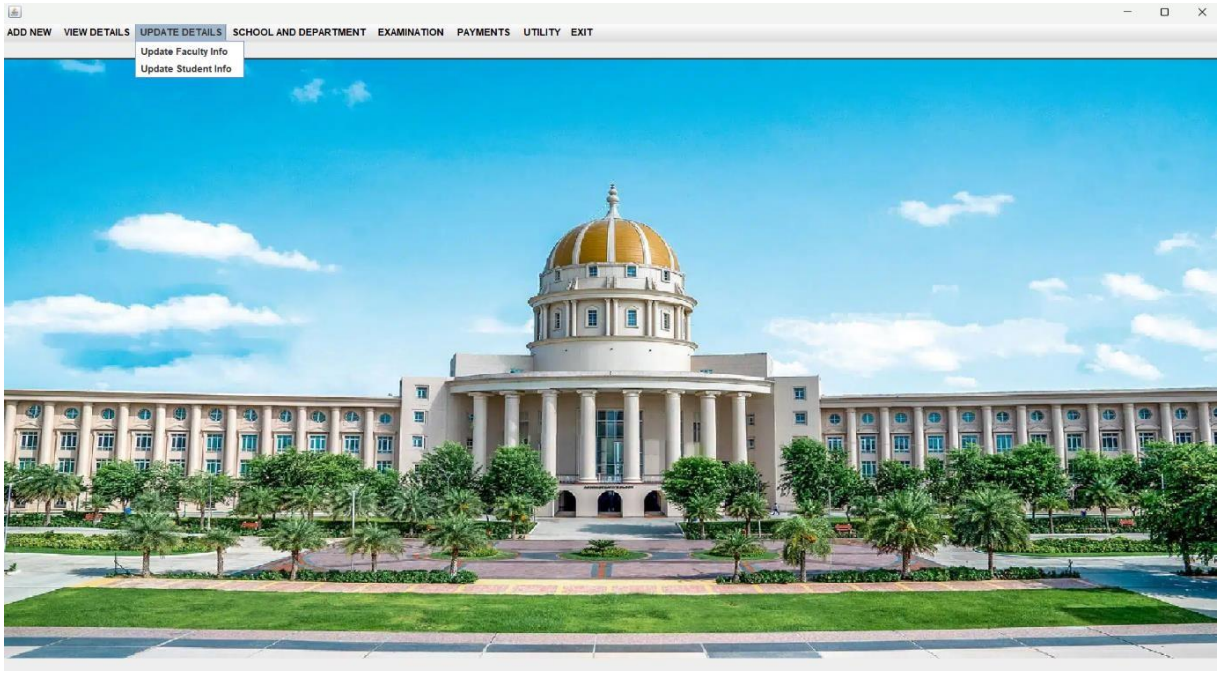
Connection to Database: Upon user submission of credentials, the LOGIN page establishes a connection to the underlying database using Java's JDBC (Java Database Connectivity) API. This allows the system to query the database for user authentication purposes, verifying whether the entered credentials match those stored in the database. Successful authentication grants access to the system, while unsuccessful attempts prompt appropriate error messages.

LOGIN CODE(java):

```
package university.manangement.system;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.ResultSet;

public class login extends JFrame implements ActionListener {
    JTextField textFieldName;
    JPasswordField passwordField;
    JButton login, back;
    login(){

        JLabel heading = new JLabel("Admin Portal");
        heading.setBounds(90,10,200,50);
        heading.setFont(new Font("serif",Font.BOLD,30));
        add(heading);

        JLabel labelName  = new JLabel("Username");
        labelName.setBounds(40,70,100,20);
        add(labelName);

        textFieldName = new JTextField();
        textFieldName.setBounds(150,70,150,20);
        add(textFieldName);

        JLabel labelPass = new JLabel("Password");
        labelPass.setBounds(40,110,100,20);
        add(labelPass);

        passwordField = new JPasswordField();
        passwordField.setBounds(150,110,150,20);
        add(passwordField);

        login = new JButton("LOGIN");
        login.setBounds(40,160,120,30);
        login.setBackground(Color.BLACK);
        login.setForeground(Color.WHITE);
        login.addActionListener(this);
        add(login);

        back = new JButton("BACK");
        back.setBounds(180,160,120,30);
        back.setBackground(Color.BLACK);
        back.setForeground(Color.WHITE);
        back.addActionListener(this);
        add(back);

        ImageIcon I1 = new
ImageIcon(ClassLoader.getSystemResource("icon/second.png"));
        Image I2 =
I1.getImage().getScaledInstance(200,200,Image.SCALE_DEFAULT);
        ImageIcon I3 = new ImageIcon(I2);
        JLabel img = new JLabel(I3);
        img.setBounds(350,20,200,200);
        add(img);
```

```
        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("icon/loginback.png"));
        Image i2 =
i1.getImage().getScaledInstance(600,300,Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel IMG = new JLabel(i3);
        IMG.setBounds(0,0,600,300);
        add(IMG);

        setSize(600,300);
        setLocation(500,250);
        setLayout(null);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == login){
            String username = textFieldName.getText();
            String password = passwordField.getText();
            String query = "select * from login where
username='"+username+"' and password='"+password+"'";
            try{
                conn c = new conn();
                ResultSet resultset = c.statement.executeQuery(query);
                if(resultset.next()){
                    setVisible(false);
                    new main_class();

                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Invalid username
or password");

                }
            }catch (Exception E){
                E.printStackTrace();
            }
        }else{
            setVisible(false);
        }
    }

    public static void main(String[] args) {
        new login();
    }
}
```

ADMIN DASHBOARD SCREEN



The Admin Dashboard Screen serves as the central hub of the University Management System, providing users with access to essential details and functionalities related to teachers, students, departments, examinations, and fees. Implemented using Java and Java Frames, the AD Screen offers a user-friendly interface for navigating and managing various aspects of the system. Here's a comprehensive overview:

Dashboard Layout: The Main Screen features a dashboard layout that organizes information into distinct sections for teachers, students, departments, examinations, and fees. Each section displays summary information, such as total counts or recent activities, providing users with an overview of the system's current status.

Navigation Menu: A navigation menu is incorporated into the Main Screen, allowing users to easily switch between different sections of the system. Java Frames enable the implementation of a menu bar or sidebar with intuitive navigation options, ensuring seamless access to teacher, student, department, examination, and fee details.

Teacher Details: The Main Screen includes a dedicated section for viewing and managing teacher details. Users can access a list of teachers, along with relevant information such as names, contact details, and courses taught. An update menu allows users to add new teachers, edit existing details, or delete entries as needed.

Student Details: Similarly, the Main Screen provides a section for viewing and managing student details. Users can access a list of students, including personal information, academic

records, and contact details. The update menu allows users to add new students, update existing information, or remove entries from the system.

Department Details: The Main Screen includes a section for departmental information, displaying details such as department names, heads, and courses offered. Users can utilize the update menu to add new departments, edit existing details, or delete departments as necessary.

Examination Details: An examination section on the Main Screen allows users to view details of examinations conducted within the university. Information such as exam dates, courses, and marks obtained by students is presented. The update menu enables users to add new examination records, update existing information, or remove entries from the system.

Fee Details: Finally, the Main Screen includes a section for managing fee details related to student payments. Users can view fee records, including student names, amounts, and payment statuses. The update menu facilitates the addition of new fee records, updates to existing payments, and deletion of obsolete entries.

Interactive Features: Java Frames allow for the incorporation of interactive features within the Main Screen, such as clickable buttons, dropdown menus, and input fields. Users can interact with these elements to perform actions such as adding new records, updating information, or navigating to different sections of the system.

Overall, the Main Screen serves as a comprehensive interface for managing teacher, student, department, examination, and fee details within the University Management System. Through its intuitive layout, navigation menu, and update functionality, users can efficiently interact with the system and perform essential tasks with ease.

Admin Dashboard screen code:

```
package university.manangement.system;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;



public class main_class extends JFrame implements ActionListener {
    main_class(){
        ImageIcon I1 = new
ImageIcon(ClassLoader.getSystemResource("icon/muj.png"));
        Image I2 =
I1.getImage().getScaledInstance(1540,750,Image.SCALE_DEFAULT);
        ImageIcon I3 = new ImageIcon(I2);
        JLabel img = new JLabel(I3);
        add(img);

        JMenuBar mb = new JMenuBar();

        JMenu newInfo = new JMenu("ADD NEW ");
        newInfo.setForeground(Color.BLACK);
        mb.add(newInfo);

        JMenuItem faculty_info = new JMenuItem("Add New Faculty");
        faculty_info.setBackground(Color.WHITE);
        faculty_info.addActionListener(this);
        newInfo.add(faculty_info);

        JMenuItem student_info = new JMenuItem("Add New Student");
        student_info.setBackground(Color.WHITE);
        student_info.addActionListener(this);
        newInfo.add(student_info);

        JMenu viewDetails = new JMenu("VIEW DETAILS");
        viewDetails.setForeground(Color.BLACK);
        mb.add(viewDetails);

        JMenuItem faculty_det = new JMenuItem("Faculty Details");
        faculty_det.setBackground(Color.WHITE);
        faculty_det.addActionListener(this);
        viewDetails.add(faculty_det);

        JMenuItem student_det = new JMenuItem("Student Details");
        student_det.setBackground(Color.WHITE);
        student_det.addActionListener(this);
        viewDetails.add(student_det);



        JMenu update = new JMenu("UPDATE DETAILS");
        update.setForeground(Color.BLACK);
        mb.add(update);

        JMenuItem faculty_up = new JMenuItem("Update Faculty Info");
        faculty_up.setBackground(Color.WHITE);
        faculty_up.addActionListener(this);
        update.add(faculty_up);
```

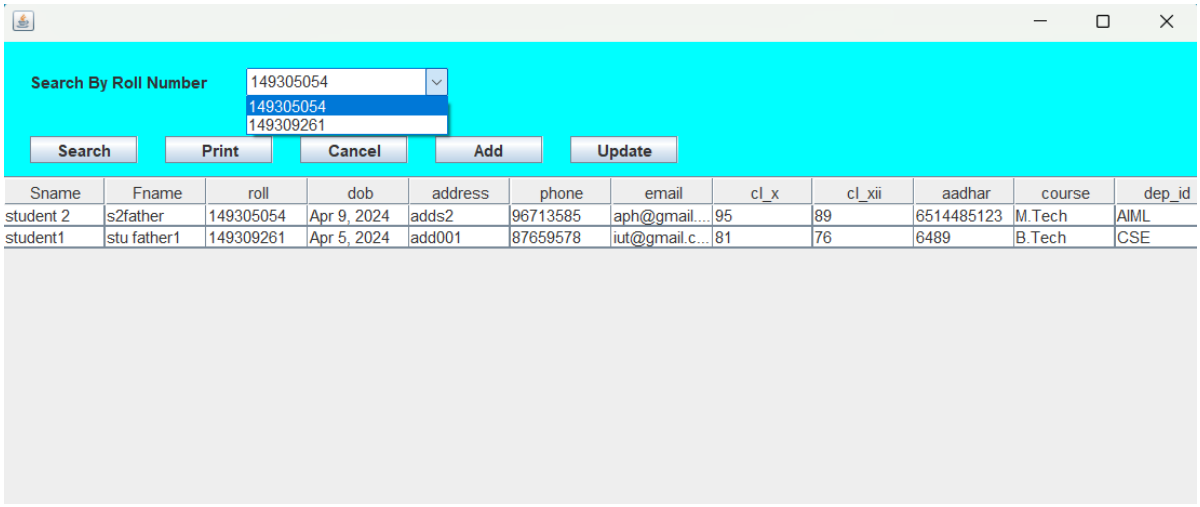
```
        JMenuItem student_up = new JMenuItem("Update Student Info");
        student_up.setBackground(Color.WHITE);
        student_up.addActionListener(this);
        update.add(student_up);

        JMenu scdep = new JMenu("SCHOOL AND DEPARTMENT ");
        scdep.setForeground(Color.BLACK);
        mb.add(scdep);

        JMenuItem school_info = new JMenuItem("View School Details");
        school_info.setBackground(Color.WHITE);
        school_info.addActionListener(this);
        scdep.add(school_info);

        JMenuItem dep_info = new JMenuItem("View Department Details");
        dep_info.setBackground(Color.WHITE);
        dep_info.addActionListener(this);
        scdep.add(dep_info);

        JMenu exm = new JMenu("EXAMINATION ");
        exm.setForeground(Color.BLACK);
        mb.add(exm);

        JMenuItem mrk = new JMenuItem("Enter Marks");
        mrk.setBackground(Color.WHITE);
        mrk.addActionListener(this);
        exm.add(mrk);

        JMenuItem mrk_det = new JMenuItem("View Result");
        mrk_det.setBackground(Color.WHITE);
        mrk_det.addActionListener(this);
        exm.add(mrk_det);

        JMenu pay = new JMenu("PAYMENTS ");
        pay.setForeground(Color.BLACK);
        mb.add(pay);

        JMenuItem fs = new JMenuItem("Fee Structure");
        fs.setBackground(Color.WHITE);
        fs.addActionListener(this);
        pay.add(fs);

        JMenuItem pfe = new JMenuItem("Pay Fee");
        pfe.setBackground(Color.WHITE);
        pfe.addActionListener(this);
        pay.add(pfe);

        JMenu utility = new JMenu("UTILITY");
        utility.setForeground(Color.BLACK);
        mb.add(utility);

        JMenuItem calc = new JMenuItem("Calculator");
        calc.setBackground(Color.WHITE);
        calc.addActionListener(this);
        utility.add(calc);

        JMenuItem cale = new JMenuItem("Calendar");
        cale.setBackground(Color.WHITE);
        cale.addActionListener(this);
        utility.add(cale);
```

```
        JMenuItem notepad = new JMenuItem("Notepad");
        notepad.setBackground(Color.WHITE);
        notepad.addActionListener(this);
        utility.add(notepad);


        JMenu exit = new JMenu("EXIT");
        exit.setForeground(Color.BLACK);
        mb.add(exit);

        JMenuItem Exit = new JMenuItem("Exit");
        Exit.setBackground(Color.WHITE);
        Exit.addActionListener(this);
        exit.add(Exit);







        setJMenuBar(mb);

        setSize(1540,850);
       setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String sm = e.getActionCommand();
        if(sm.equals("Exit")){
            System.exit(15);
        } else if(sm.equalsIgnoreCase("Calculator")){
            try{
                Runtime.getRuntime().exec("calc.exe");
            }catch (Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("notepad")){
            try{
                Runtime.getRuntime().exec("notepad.exe");
            } catch (Exception E) {
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Faculty Details")){
            try{
                new teacherDet();
            } catch (Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Enter Marks")){
            try{
                new enter_marks();
            } catch (Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("View Result")){
            try{
                new mark_det();
            } catch (Exception E){
```

```
                E.printStackTrace();
            }
        }  else if(sm.equalsIgnoreCase("Pay Fee")){
            try{
                new StudentFeeForm();
            } catch (Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Fee Structure")){
            try{
                new Fee_Structure();
            } catch (Exception E){
                E.printStackTrace();
            }
        }else if(sm.equalsIgnoreCase("Student Details")){
            try{
                new StudentDet();
            } catch (Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Calendar")){
            try{

                new cale();

            } catch(Exception E){
                E.printStackTrace();
            }
        }else if(sm.equalsIgnoreCase("Add New Faculty")){
            try{

                new Add_Faculty();

            } catch(Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Add New Student")){
            try{

                new add_student();

            } catch(Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("View School Details")){
            try{

                  new school_det();

            } catch(Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("View Department Details")){
            try{

               new dep_det();

            } catch(Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Update Student Info")){
```

```
            try{
                new update_stud();
            } catch(Exception E){
                E.printStackTrace();
            }
        } else if(sm.equalsIgnoreCase("Update Faculty Info")){
            try{
                new update_fac();
            } catch(Exception E){
                E.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        new main_class();
    }
}
```

STUDENT DETAILS SECTION



The Student Details Section in the University Management System plays a crucial role in managing student information efficiently. Implemented using Java and Java Frames, this section provides a user-friendly interface for viewing, updating, and managing student details, including roll number, name, Aadhar number, address, and department affiliation. Here's an in-depth explanation of its features and functionalities:

Display of Student Information: The Student Details Section presents a comprehensive list of student records, displaying essential information such as roll number, name, Aadhar number, address, and department affiliation. Java Frames enable the creation of a table or list view that organizes student details in a clear and structured format for easy reference.

Roll Number and Name: Each student record prominently displays the roll number and name, serving as unique identifiers for individual students. Users can quickly locate specific students by searching or sorting the records based on roll numbers or names, enhancing accessibility and navigation within the system.

Aadhar Number and Address: Additional details such as Aadhar number and address provide comprehensive information about each student. These fields contribute to accurate record-keeping and facilitate communication with students regarding academic matters, administrative updates, or emergency situations.

Department Affiliation: The Student Details Section includes information about each student's department affiliation. This allows users to associate students with their respective academic departments, enabling efficient organization and management of student records based on departmental categorization.

Update Menu: An update menu integrated into the Student Details Section offers users the ability to perform various actions on student records seamlessly. Using Java Frames, the update menu provides options for adding new student records, editing existing information, and deleting obsolete entries from the system.

Add New Student: Users can utilize the update menu to add new student records to the system. This functionality prompts users to enter relevant details such as roll number, name, Aadhar number, address, and department affiliation. Data validation techniques ensure the accuracy and completeness of user inputs before adding new records to the database.

Edit Existing Information: The update menu enables users to update or modify existing student information as needed. By selecting a student record from the list, users can edit specific fields such as name, Aadhar number, address, or department affiliation. Changes made through the update menu are immediately reflected in the database, ensuring data consistency and integrity.

Data Validation and Error Handling: The Student Details Section implements robust data validation mechanisms to ensure the accuracy and integrity of student information. Java Frames facilitate the validation of user inputs, such as Aadhar numbers and addresses, to prevent errors and inconsistencies. Additionally, error handling features prompt users with informative messages in case of invalid inputs or system errors, guiding them to correct any issues encountered during data entry or updates.

In summary, the Student Details Section, implemented using Java and Java Frames, offers a comprehensive and user-friendly interface for managing student information in the University Management System. Through its display of essential student details and integration of an update menu, users can efficiently view, update, and manage student records, contributing to

streamlined administrative processes and enhanced data management capabilities within the system.

Student Details Code:

```
package university.manangement.system;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.print.PrinterException;
import java.sql.ResultSet;
import net.proteanit.sql.DbUtils;

public class StudentDet extends JFrame implements ActionListener {
    Choice choice;
    JTable table;
    JButton search, print, update, add, cancel;
    StudentDet(){
        getContentPane().setBackground(Color.CYAN);

        JLabel heading = new JLabel("Search By Roll Number");
        heading.setBounds(20,20,150,20);
        add(heading);

        choice = new Choice();
        choice.setBounds(180,20,150,20);
        add(choice);
        try{
            conn c = new conn();
            ResultSet resultSet= c.statement.executeQuery("select * from
student");
            while(resultSet.next()){
                choice.add(resultSet.getString("roll"));
            }
        }catch(Exception e){
            e.printStackTrace();
        }

        table = new JTable();
        try{
            conn c = new conn();
            ResultSet resultSet = c.statement.executeQuery("select * from
student");
            table.setModel(DbUtils.resultSetToTableModel(resultSet));
        }catch(Exception e){
            e.printStackTrace();
        }
        JScrollPane js = new JScrollPane(table);
        js.setBounds(0,100,900,600);
        add(js);

        search = new JButton("Search");
        search.setBounds(20,70,80,20);
        search.addActionListener(this);
        add(search);

        print = new JButton("Print");
        print.setBounds(120,70,80,20);
```

```
        print.addActionListener(this);
        add(print);

        cancel = new JButton("Cancel");
        cancel.setBounds(220,70,80,20);
        cancel.addActionListener(this);
        add(cancel);

        add = new JButton("Add");
        add.setBounds(320,70,80,20);
        add.addActionListener(this);
        add(add);

        update = new JButton("Update");
        update.setBounds(420,70,80,20);
        update.addActionListener(this);
        add(update);

        setLayout(null);
        setSize(900,700);
        setLocation(300,100);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==search){
            String q = "select * from student where roll =
'"+choice.getSelectedItem()+"'";
            try{
                conn c = new conn();
                ResultSet resultSet = c.statement.executeQuery(q);
                table.setModel(DbUtils.resultSetToTableModel(resultSet));


            }catch(Exception E){
                E.printStackTrace();
            }
        } else if(e.getSource()== print){
            try{
                table.print();
            } catch (PrinterException ex) {
                throw new RuntimeException(ex);
            }
        } else if(e.getSource() == add){
            setVisible(false);
            new add_student();
        } else if(e.getSource() == update){
            new update_stud();
        } else {
            setVisible(false);
        }
    }

    public static void main(String[] args) {
        new StudentDet();
    }
}
```

DATABASE CONNECTIVITY

```java
package university.manangement.system;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class conn {
    2 usages
    Connection connection;
    37 usages
    Statement statement;

    34 usages
    conn(){
        try{
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection( url: "jdbc:mysql:///universitymanagement", user: "root",  password: "root");
            statement = connection.createStatement();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

The following code is written in JAVA with the use of IntelliJ IDEA. The sole purpose of this code is to connect smoothly to the MySQL Database with the help of JDBC Connectivity.

Here's an overview of how JDBC is utilized for database connectivity within the project:

Driver Registration: The first step in JDBC database connectivity is registering the appropriate database driver. This involves loading the JDBC driver class provided by the database vendor into the Java application's classpath. For example, if the project uses MySQL as the RDBMS, the MySQL JDBC driver (e.g., com.mysql.cj.jdbc.Driver) is registered using Class.forName().

Connection Establishment: Once the driver is registered, the Java application establishes a connection to the database using the DriverManager.getConnection() method. This method requires parameters such as the database URL, username, and password to connect to the database server. For instance, a MySQL database connection string might resemble: jdbc:mysql://localhost:3306/database_name.

Executing SQL Statements: With the database connection established, the Java application can execute SQL statements against the database using java.sql.Statement or java.sql.PreparedStatement objects. These statements can include queries (SELECT), data manipulation operations (INSERT, UPDATE, DELETE), or even data definition commands (CREATE, ALTER, DROP).

FUTURE ADDONS IN THE PROJECT

Several possible future addons can enhance the functionality, usability, and efficiency of the University Management System project. Here are some ideas:

1. **User Roles and Permissions**: Implement role-based access control (RBAC) to differentiate between administrators, teachers, and students. Define specific permissions for each role to restrict access to sensitive functionalities or data.

2. **Attendance Tracking**: Introduce a feature for recording and monitoring student attendance. Teachers can mark attendance for their classes, and administrators can generate attendance reports for analysis and intervention.

3. **Course Management**: Enhance course management capabilities by allowing administrators to create, modify, and delete course offerings. Include features for assigning teachers to courses and managing course schedules.

4. **Gradebook**: Develop a gradebook module to enable teachers to record and manage student grades for assignments, quizzes, and exams. Students can access their grades and monitor their academic progress.

5. **Messaging System**: Implement an internal messaging system for communication between administrators, teachers, and students. Allow users to send messages, announcements, and notifications within the system.

6. **Document Management**: Introduce a document management system for uploading, organizing, and sharing academic resources such as lecture notes, presentations, and study materials.

7. **Analytics and Reporting**: Incorporate analytics tools to analyze student performance, attendance trends, and course effectiveness. Generate reports and visualizations to aid decision-making and improve academic outcomes.

8. **Online Assessments**: Develop functionality for conducting online assessments, quizzes, and examinations within the system. Include features for question creation, test scheduling, and automatic grading.

9. **Integration with Learning Management Systems (LMS)**: Integrate the University Management System with popular LMS platforms to leverage additional educational resources, tools, and functionalities.

10. **Mobile Application**: Develop a mobile application version of the system to provide users with convenient access to essential features and information on their smartphones or tablets.

By incorporating these future addons, the University Management System can evolve into a comprehensive and versatile platform that meets the evolving needs of administrators, teachers, and students, ultimately enhancing the overall educational experience and efficiency of university operations.

LIST OF TABLES

LIST OF FIGURES

BIBLIOGRAPHY

- DRAW.IO for the ER Diagram
- https://www.youtube.com/playlist?list=PLbKub4Jss9oVR0Uo9fuMFBcVsplNnSzD2
- www.Wikipedia.com

At last, I would like to thank Dr. Sourabh Singh Verma sir for continuous guidance and all my teachers and friends who helped in successful completion of the project.