**1. PROJECT ABSTRACT**

This project aims to create a Java-based application that allows users to collect and store data on students and teachers in a database. The application will have a user-friendly interface that allows users to input and manage student and teacher information, including personal details, contact information, and academic performance. The collected data will be stored in a database for easy retrieval and analysis. The project will utilize various Java technologies such as JDBC for database connectivity and JSP for creating the user interface. The end goal of this project is to provide a reliable and efficient tool for managing and organizing student and teacher information.

**1.1 INTRODUCTION TO SCHOOL MANAGEMENT SYSTEM**

A school management system is a software application that aims to automate and streamline the various administrative tasks and processes of a school. These tasks may include student registration, attendance tracking, grade management, and communication with parents and teachers. A Java-based school management system will provide a reliable and efficient tool for managing and organizing student and teacher information, as well as streamlining various school-related tasks. This project aims to create such a system, utilizing various Java technologies such as JDBC for database connectivity and JSP for creating the user interface. The system will have a user-friendly interface, making it easy for teachers, administrators, and staff to input and access student and teacher information. The end goal of this project is to provide a comprehensive and efficient solution for managing and organizing school-related tasks and information.

**1.2 PURPOSE OF PROJECT**

The aim of this school management system project is to develop a comprehensive and efficient solution that automates and streamlines various administrative tasks and processes of a school, utilizing Java technologies, to provide an easy way to manage and organize student and teacher information.

**OLD METHODS OF SCHOOL MANAGEMENT SYSTEM**

The existing school management systems are mostly based on manual processes and paper-based documentation. These systems can be time-consuming, prone to errors, and lack the ability to generate reports and analyze data. The existing systems also face a challenge in terms of communication between teachers, students and parents, which is a crucial aspect of school management. With the use of Java-based technology, a new school management system can be developed that can automate and streamline these processes, providing a more efficient and effective solution for managing and organizing school-related tasks and information.

**2. SYSTEM REQUIRMENTS**

**2.1 SOFTWARE REQUIREMENTS**

* Operating System : Windows 11
* Programming language : Java
* IDE : Apache NetBeans IDE 16
* Database : MySQL

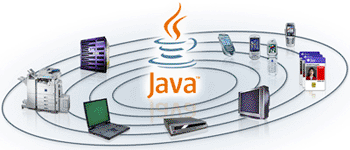
**2.2 HARDWARE REQUIREMENTS**

* Processor : Intel core i3 and above
* RAM : 4 GB(Min)
* Hard Disk : 128 GB
* Key Board : Standard Windows Keyboard
* Mouse : Touchpad or Optical Mouse

**3. SOFTWARES USED FOR PROJECT DEVELOPMENT**

**3.1 INTRODUCTION TO JAVA DEVELOPMENT**

The Java platform consists of the Java application programming interfaces (APIs) and the Java Virtual Machine (JVM)



When the chronicle of computer languages is written, the following will be said: B led to C, C evolved into C++, and C++ set the stage for Java. To understand the reasons that drove the creation, the forces that shaped it, and the legacy that it inherits. Like the successful compute languages that came before, Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique environment.

According to Grady Booch, a leading exponent of the object-oriented approach, an object has the following characteristics.

1. It has state.
2. It may display a behavior.
3. It has a unique identity.

### The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

Simple

Architecture neutral

Object oriented

Portable

Distributed

High performance

Interpreted

Multithreaded

Robust

Dynamic

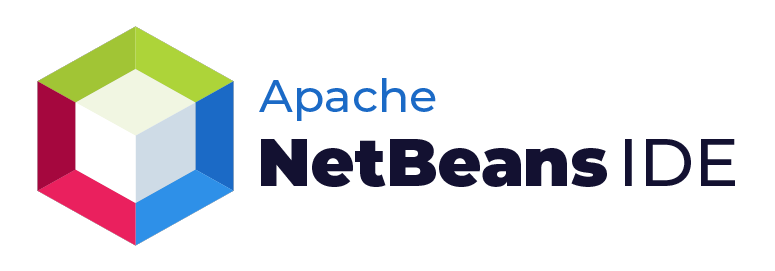
Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figureillustrates how this works.



**3.2 APACHE NETBEANS IDE 16**

NetBeans IDE is a free and open-source integrated development environment (IDE) for developing Java applications. It is written in Java and runs on multiple platforms, including Windows, Mac, and Linux. It provides a wide range of features for developing, debugging, and deploying Java applications, including code completion, error checking, and integrated debugging and testing tools. NetBeans IDE also supports a wide range of other languages, including C, C++, PHP, and HTML5. It also supports other technologies like JavaFX, Spring Framework, and more.



**MySQL DATABASE**

MySQL is a free, open-source relational database management system (RDBMS) that uses the Structured Query Language (SQL) for managing data. It is widely used for managing data in web-based applications and is a popular choice for data management in small and large scale applications. MySQL is known for its reliability, robustness, and ease of use. It also supports a wide range of programming languages such as Java, C++, C#, and Python. It also has a variety of options for data replication and backup, making it suitable for high-availability scenarios. Furthermore, it is a part of LAMP (Linux, Apache, MySQL, PHP/Perl/Python) stack which is a popular solution for building web-based application.



**What is a user interface**

The user interface is that part of a program that interacts with the user of the program. User interfaces take many forms. These forms range in complexity from simple command-line interfaces to the point-and-click graphical user interfaces provided by many modern applications.

At the lowest level, the operating system transmits information from the mouse and keyboard to the program as input, and provides pixels for program output. The AWT was designed so that programmers don't have worry about the details of tracking the mouse or reading the keyboard, nor attend to the details of writing to the screen. The AWT provides a well-designed object-oriented interface to these low-level services and resources.

Because the Java programming language is platform-independent, the AWT must also be platform-independent. The AWT was designed to provide a common set of tools for graphical user interface design that work on a variety of platforms. The user interface elements provided by the AWT are implemented using each platform's native GUI toolkit, thereby preserving the look and feel of each platform. This is one of the AWT's strongest points. The disadvantage of such an approach is the fact that a graphical user interface designed on one platform may look different when displayed on another platform.

**Components and containers**

A graphical user interface is built of graphical elements called components. Typical components include such items as buttons, scrollbars, and text fields. Components allow the user to interact with the program and provide the user with visual feedback about the state of the program. In the AWT, all user interface components are instances of class Component or one of its subtypes.

Components do not stand alone, but rather are found within containers. Containers contain and control the layout of components. Containers are themselves components, and can thus be placed inside other containers. In the AWT, all containers are instances of class Container or one of its subtypes.

Spatially, components must fit completely within the container that contains them. This nesting of components (including containers) into containers creates a tree of elements, starting with the container at the root of the tree and expanding out to the leaves, which are components such as buttons.

The illustration in Figure 1 depicts a simple graphical user interface as it would look when displayed under Windows 95. Figure 2 shows the interface components from Figure 1 arranged as a tree.

**Types of components**

Figure 3 shows the inheritance relationship between the user interface component classes provided by the AWT. Class Component defines the interface to which all components must adhere.

The AWT provides nine basic non-container component classes from which a user interface may be constructed. (Of course, new component classes may be derived from any of these or from class Component itself.) These nine classes are class Button, Canvas, Checkbox, Choice, Label, List, Scrollbar, TextArea, and TextField. Figure 4 depicts an instance of each class.

**Types of containers**

The AWT provides four container classes. They are class Window and its two subtypes -- class Frame and class Dialog -- as well as the Panel class. In addition to the containers provided by the AWT, the Applet class is a container -- it is a subtype of the Panel class and can therefore hold components. Brief descriptions of each container class provided by the AWT are provided below.

|  |  |
| --- | --- |
| Window | A top-level display surface (a window). An instance of the Window class is not attached to nor embedded within another container. An instance of the Window class has no border and no title. |
| Frame | A top-level display surface (a window) with a border and title. An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class. |
| Dialog | A top-level display surface (a window) with a border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class. |
| Panel | A generic container for holding components. An instance of the Panel class provides a container to which to add components. |

Popular on Java Word.

**Creating a container**

Before adding the components that make up a user interface, the programmer must create a container. When building an application, the programmer must first create an instance of class Window or class Frame. When building an applet, a frame (the browser window) already exists. Since the Applet class is a subtype of the Panel class, the programmer can add the components to the instance of the Applet class itself.

The code in Listing 1 creates an empty frame. The title of the frame ("Example 1") is set in the call to the constructor. A frame is initially invisible and must be made visible by invoking its show() method.

importjava.awt.\*;

public class Example1

{

public static void main(String [] args)

{

Frame f = new Frame("Example 1");

f.show();

}

}

**Listing 1.**

*An empty frame*

The code in Listing 2 extends the code from Listing 1 so that the new class inherits from class Panel. In the main() method, an instance of this new class is created and added to the Frame object via a call to the add() method. The result is then displayed. The results of both examples should look identical (that is, they should look quite uninteresting).

importjava.awt.\*;

public class Example1a extends Panel

{

public static void main(String [] args)

{

Frame f = new Frame("Example 1a");

Example1a ex = new Example1a();

f.add("Center", ex);

f.pack();

f.show();

}

}

**Listing 2.**

*A frame with an empty panel*

By deriving the new class from class Applet instead of class Panel, this example can now run as either a standalone application or as an applet embedded in a Web page. The code for this example is provided in Listing 3. The resulting applet is displayed in Figure 5 (and is still quite uninteresting).

importjava.awt.\*;

public class Example1b extends java.applet.Applet

{

public static void main(String [] args)

{

Frame f = new Frame("Example 1b");

Example1b ex = new Example1b();

f.add("Center", ex);

f.pack();

f.show();

}

}

**3.3 SDLC LIFE CYCLE (WATERFALL MODEL)**

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development life cycle consists of the following stages:

**Stages of SDLC:**

* Requirement Gathering and Analysis
* Designing
* Coding
* Testing
* Deployment

**Stage 1: Planning and Requirement Analysis**

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

**Stage 2: Defining Requirements**

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through .SRS. . Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

**Stage 3: Designing the product architecture**

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

**Stage 4: Building or Developing the Product**

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

**Stage 5: Testing the Product**

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

**Stage 6: Deployment in the Market and Maintenance**

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations. business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

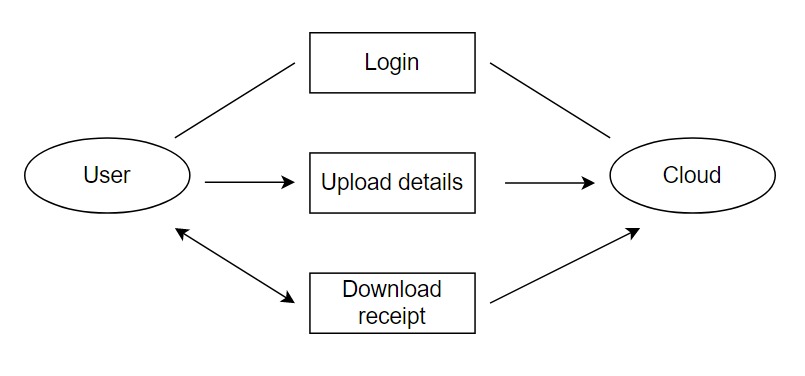
Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base maintenance is done for the existing customer base.

**USE CASE DIAGRAM**

Use Case Model is an approach that is a combination of text and pictures in order to improve the understanding of requirements. A use case model is describes the complete functionality of a system by identifying how everything that is outside the system interacts with it. A Use Case Diagram is given below that relates to this application.

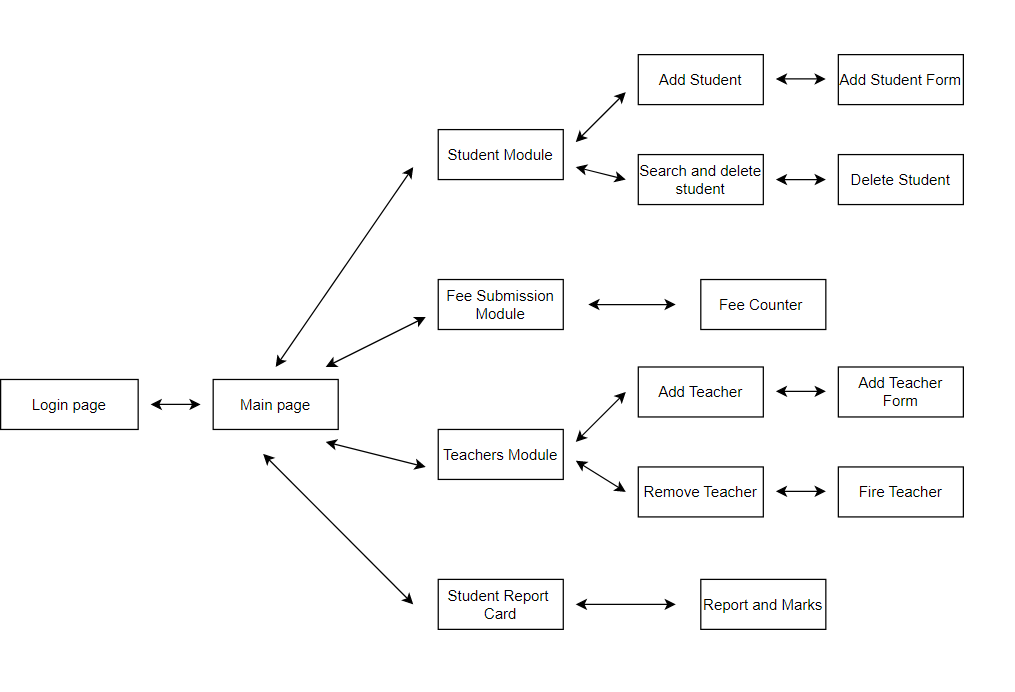
**Description** – This project is a Java application that manages a system of files.

**Actors** – User and Cloud

****

**SEQUENCE DIAGRAM**

A sequence diagram is a type of UML diagram that illustrates the interactions between objects and classes in a system, over time. A sequence diagram for a school management system project to show the flow of control within the system.



**4.SAMPLE CODING**

package schoolmanagementsystem;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.DriverManager;

import java.sql.ResultSet;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author dell

\*/

public class LoginPage extends javax.swing.JFrame {

/\*\*

\* Creates new form LoginPage

\*/

public LoginPage() {

initComponents();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

jLabel3 = new javax.swing.JLabel();

jLabel4 = new javax.swing.JLabel();

jLabel5 = new javax.swing.JLabel();

user = new javax.swing.JTextField();

pass = new javax.swing.JPasswordField();

jLabel6 = new javax.swing.JLabel();

sc = new javax.swing.JTextField();

jButton1 = new javax.swing.JButton();

jLabel7 = new javax.swing.JLabel();

jButton2 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

jLabel1.setFont(new java.awt.Font("Times New Roman", 1, 24)); // NOI18N

jLabel1.setForeground(new java.awt.Color(153, 0, 0));

jLabel1.setText("SCHOOL MANAGEMENT SYSTEM");

getContentPane().add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(80, 170, -1, -1));

jLabel2.setFont(new java.awt.Font("Tempus Sans ITC", 1, 11)); // NOI18N

jLabel2.setText("LOGIN PAGE");

getContentPane().add(jLabel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(240, 210, -1, -1));

jLabel3.setText("Username:");

getContentPane().add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 250, -1, -1));

jLabel4.setText("Password:");

getContentPane().add(jLabel4, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 300, -1, -1));

jLabel5.setText("Forgot Password?");

getContentPane().add(jLabel5, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 350, -1, -1));

getContentPane().add(user, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 250, 201, -1));

getContentPane().add(pass, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 290, 201, -1));

jLabel6.setText("Enter Secret Code:");

getContentPane().add(jLabel6, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 390, -1, -1));

getContentPane().add(sc, new org.netbeans.lib.awtextra.AbsoluteConstraints(230, 390, 71, -1));

jButton1.setText("Submit");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

getContentPane().add(jButton1, new org.netbeans.lib.awtextra.AbsoluteConstraints(340, 330, -1, -1));

jLabel7.setIcon(new javax.swing.ImageIcon(getClass().getResource("/schoolmanagementsystem/7645189.jpg"))); // NOI18N

jLabel7.setText("jLabel7");

getContentPane().add(jLabel7, new org.netbeans.lib.awtextra.AbsoluteConstraints(174, 17, 190, 140));

jButton2.setText("submit");

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

getContentPane().add(jButton2, new org.netbeans.lib.awtextra.AbsoluteConstraints(310, 390, -1, -1));

pack();

}// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

String un= user.getText();

String p= pass.getText();

try{

Class.forName("com.mysql.jdbc.Driver");

Connection conn= (Connection) DriverManager.getConnection("jdbc:mysql://localhost/sms","root","");

Statement st= (Statement)conn.createStatement();

String sql= "select \* from user\_login";

ResultSet rs= st.executeQuery(sql);

while(rs.next()){

String username= rs.getString("username");

String Password= rs.getString("password");

if(un.equals(username) && p.equals(Password)){

new welcome().setVisible(true);

}

else

{

JOptionPane.showMessageDialog(this, "Username or Password is incorrect!");

}

}

}

catch(Exception e){

JOptionPane.showMessageDialog(null, "Error while establishing connection");

}

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

String secret= sc.getText();

if(secret.equals("mylogin@details")){

JOptionPane.showMessageDialog(this, "The username is 'charan' and password is '0000'");

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(LoginPage.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(LoginPage.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(LoginPage.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(LoginPage.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new LoginPage().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JLabel jLabel4;

private javax.swing.JLabel jLabel5;

private javax.swing.JLabel jLabel6;

private javax.swing.JLabel jLabel7;

private javax.swing.JPasswordField pass;

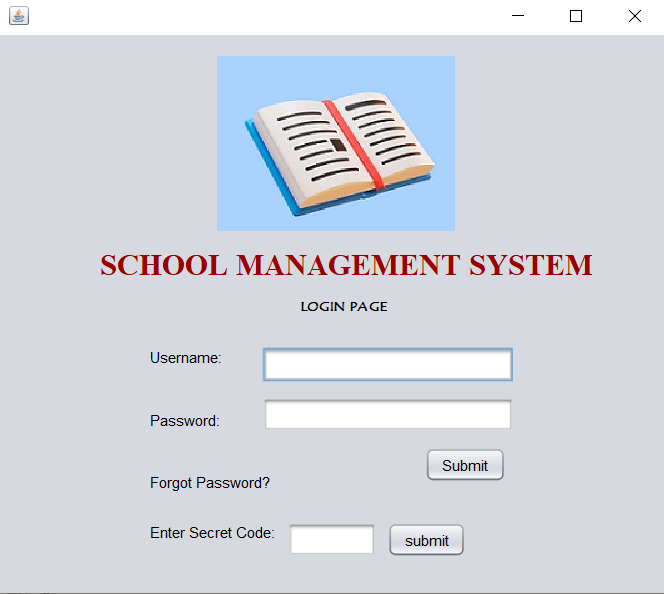
private javax.swing.JTextField sc;

private javax.swing.JTextField user;

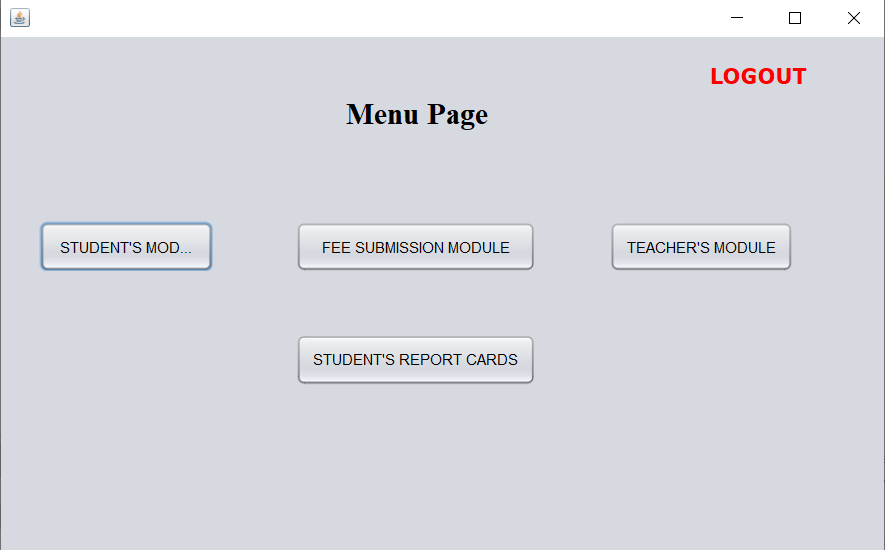
// End of variables declaration

}

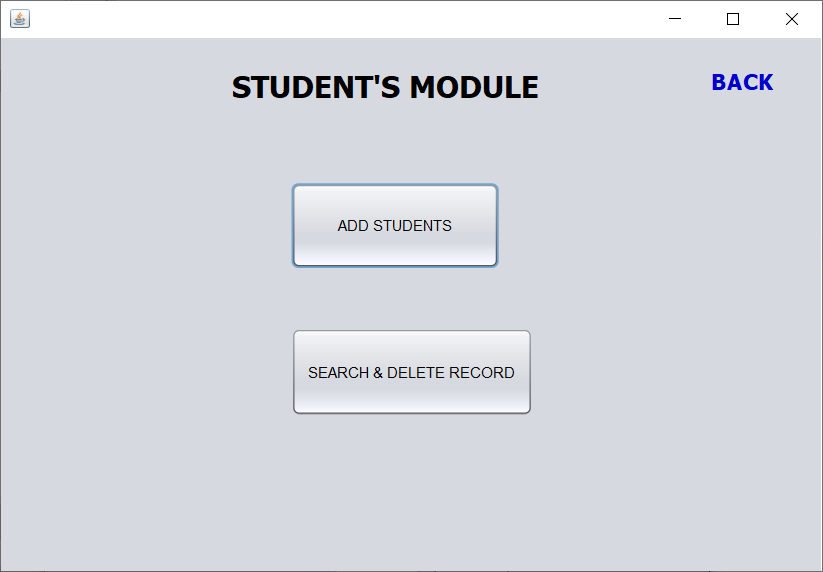
**5.OUTPUT SCREENS**

****

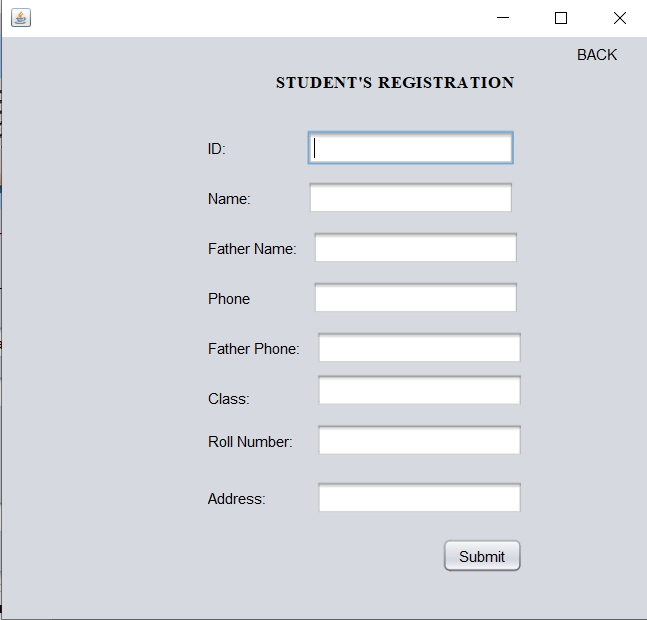
1.1 LOGIN PAGE

****

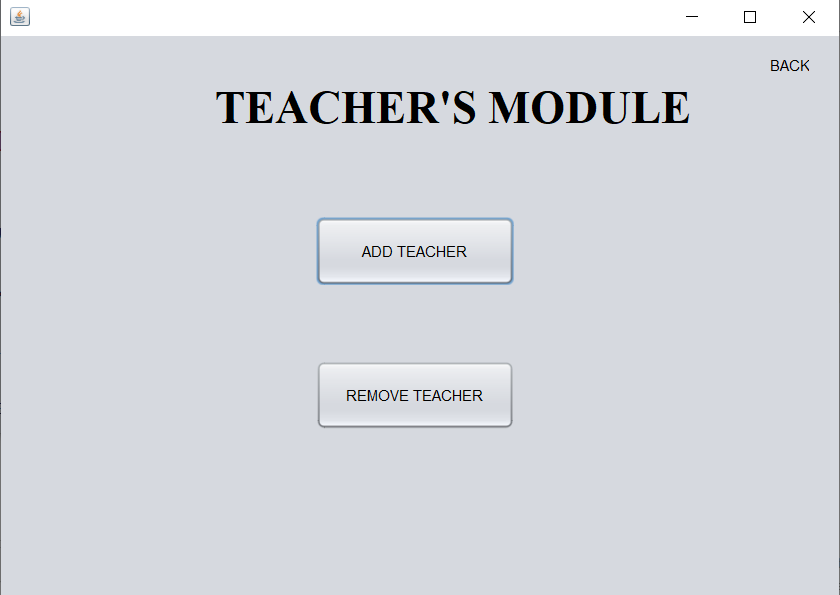
1.2 MENU PAGE



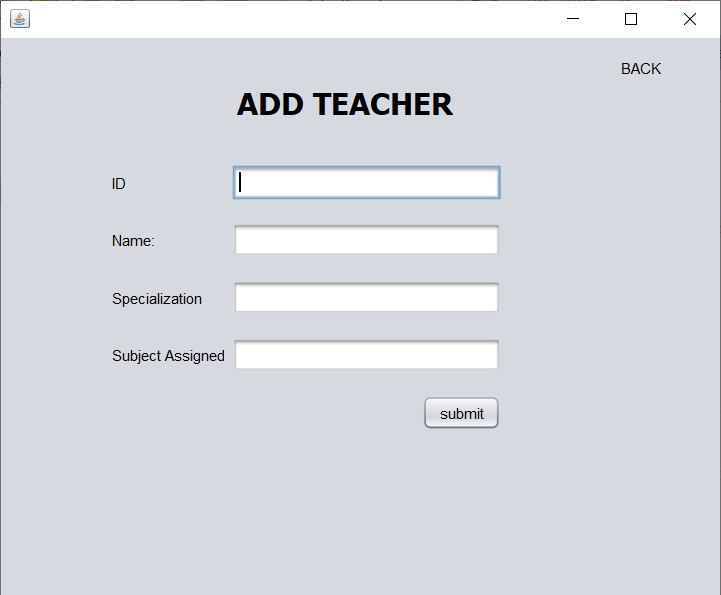
1.3 STUDENT’S MODULE PAGE



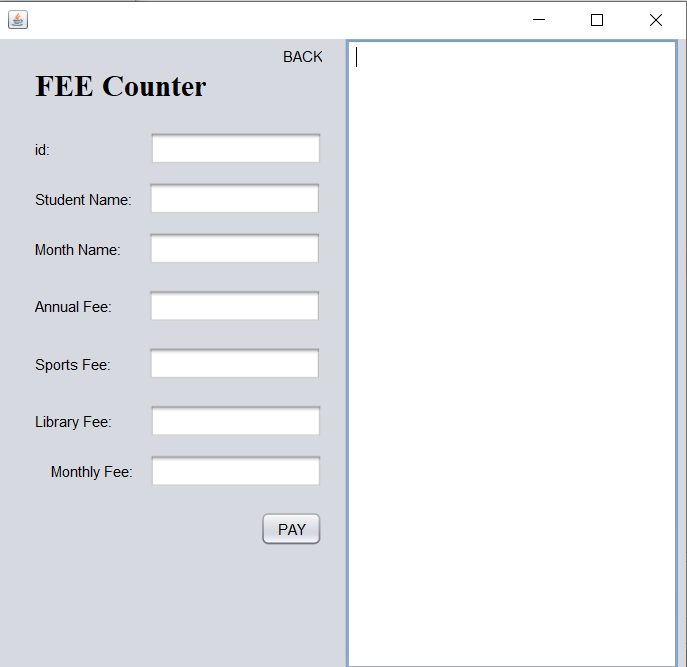
1.4 STUDENT REGISTRATION PAGE



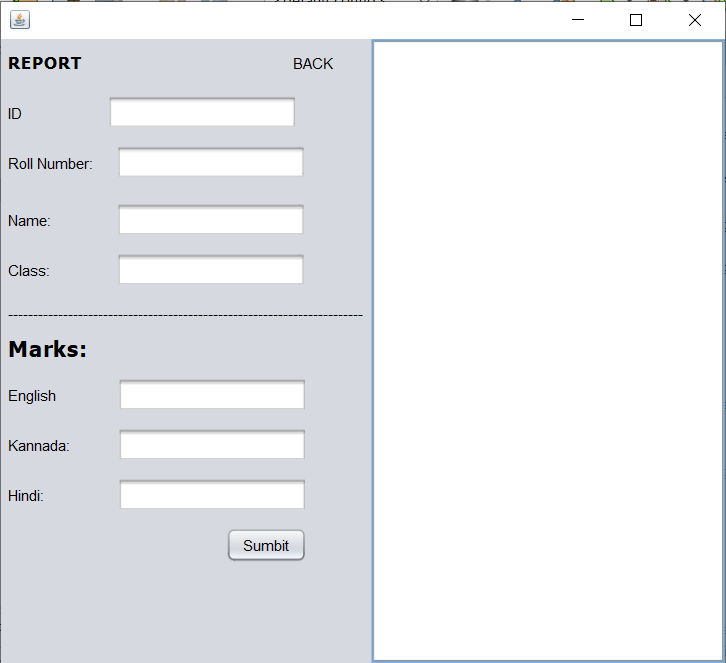
1.5 TEACHER’S MODULE PAGE



1.6 TEACHER REGISTRATION PAGE



1.7 STUDENT FEE



1.8 STUDENT REPORT CARD

**6.SYSTEM TESTING**

**INTRODUCTION TO TESTING**

System Testing involves two kinds of testing integration testing and acceptance testing. Developing a strategy for integrating the components of a software system into a functioning whole requires careful planning so that modules are available for integration when needed. Acceptance testing involves planning and execution of various tests in order to demonstrate that the implemented system satisfies the requirements document.

**6.1 Software testing**

Software Testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding, Testing presents an interesting anomaly for the software engineer.

* + 1. **Testing Objectives include**

1. Testing is a process of executing a program with the intent of finding an error
2. A good test case is one that has a probability of finding an as yet undiscovered error
3. A successful test is one that uncovers an undiscovered error
   * 1. **Testing Principles**
4. All tests should be traceable to end user requirements
5. Tests should be planned long before testing begins
6. Testing should begin on a small scale and progress towards testing in large
7. Exhaustive testing is not possible

**6.2 Testing Strategies**

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also.

**6.2.1 Psychology of Testing**

The aim of testing is often to demonstrate that a program works by showing that it has no errors. The basic purpose of testing phase is to detect the errors that may be present in the program. Hence one should not start testing with the intent of showing that a program works, but the intent should be to show that a program doesn’t work.

**6.2.2 Testing Objectives**

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say,

1. Testing is a process of executing a program with the intent of finding an error.
2. A successful test is one that uncovers an as yet undiscovered error.
3. A good test case is one that has a high probability of finding error, if it exists.
4. The tests are inadequate to detect possibly present errors.
5. The software more or less confirms to the quality and reliable standards.

**6.2.3 Levels of Testing**

In order to uncover the errors present in different phases we have the concept of levels of testing. The basic levels of testing are

**Client Needs Acceptance Testing**

Requirements System Testing

Design Integration Testing

Code Unit Testing

**Figure 8.1** Basic levels of Testing

**1. Unit testing**

Unit testing focuses verification effort on the smallest unit of software i.e. the module. Using the detailed design and the process specifications testing is done to uncover errors within the boundary of the module. All modules must be successful in the unit test before the start of the integration testing begins.

In this project each service can be thought of a module. There are so many modules like Login Module, Admin Module, and User Module. Each module has been tested by giving different sets of inputs. When developing the module as well as finishing the development so that each module works without any error. The inputs are validated when accepting from the user.

**2. Integration Testing**

After the unit testing we have to perform integration testing. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules. This testing activity can be considered as testing the design and hence the emphasis on testing module interactions.

In this project the main system is formed by integrating all the modules. When integrating all the modules I have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before Integration.

**3. System Testing**

Here the entire software system is tested. The reference document for this process is the requirements document, and the goals to see if software meets its requirements. Here entire ‘IT & Telecom Stores Management system’ has been tested against requirements of project and it is checked whether all requirements of project have been satisfied or not.

**4. Acceptance Testing**

Acceptance Test is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behavior of the system, the internal logic of program is not emphasized.

In this project **‘**IT Infrastructure Management of Database System’ I have collected some data and tested whether project is working correctly or not.

Test cases should be selected so that the largest number of attributes of an equivalence class is exercised at once. The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

**5. White Box Testing**

This is a unit testing method where a unit will be taken at a time and tested thoroughly at a statement level to find the maximum possible errors.

I tested step wise every piece of code, taking care that every statement in the code is executed at least once. The white box testing is also called Glass Box Testing.

I have generated a list of test cases, sample data. This is used to check all possible combinations of execution paths through the code at every module level.

**6. Black Box Testing**

This testing method considers a module as a single unit and checks the unit at interface and communication with other modules rather getting into details at statement level. Here the module will be treated as a block box that will take some input and generate output. Output for a given set of input combinations are forwarded to other modules.

**7.FUTURE ENHANCEMENT**

The system features can be expanded by some further improvement. The future work may include:

1. **Online Payment System:** Integrating an online payment system would enable students and parents to pay tuition fees, and other charges online, making the process more efficient and streamlined.
2. **Automated Report Cards:** An automated report card generation feature would allow teachers to easily generate report cards for their students, and for administrators to monitor student progress.
3. **Integration with other software:** Integrating the school management system with other software such as Learning Management Systems or Student Information Systems to create a more holistic and complete solution for managing student information.

These are just a few examples, and many other enhancements could be made to improve the functionality and usability of the system.

**8. CONCLUSION**

A school management system project in Java is a comprehensive and efficient solution for managing student and teacher information in an educational institution. The system is designed to automate and streamline various administrative tasks such as student registration, class scheduling, and grade management. The use of Java as the programming language ensures that the system is platform-independent and can run on a variety of devices and operating systems. The use of a relational database management system like MySQL allows for the storage and retrieval of large amounts of data in an organized and efficient manner. The system design, which includes architecture, database, user interface, class diagram, sequence diagram, flowchart and pseudocode, ensures that the final product is user-friendly and can meet the needs of the users. Overall, a school management system project in Java is a practical and effective solution for managing student and teacher information in an educational institution.

**9. BIBLIOGRAPHY**

Reference for the Project development were taken from the web sites:

1. www.google.com

2. www.youtube.com

3. www.github.com