

**COLLEGE RESOURCE MANAGEMENT SYSTEM**

**LOGIN and SIGNUP PAGE**

**A MINI PROJECT REPORT**

**Submitted by**

**MAHESHWARAN P 231501510**

**In partial fulfilment for the award of the degree of**

**BACHELOR OF TECHNOLOGY IN**

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

**RAJALAKSHMI ENGINEERING COLLEGE**

**(AUTONOMOUS) THANDALAM**

**CHENNAI-602105**

**2024 - 2025**



**BONAFIDE CERTIFICATE**

Certified that this project report **“STUDENT INFORMATION MANAGEMENT SYSTEM**” is the Bonafide work of **“MAHESHWARAN P (231501510).”** who carried out the project work under my supervision.

**Submitted for the Practical Examination held on\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

**SIGNATURE**

**Mrs. Manju S,**

**Assistant Professor (SS)**

**AIML,**

**Rajalakshmi Engineering**

**College (Autonomous),**

**Thandalam, Chennai - 602 105**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ABSTRACT**

**Abstract**

The **Student Information System** is a Java-based application designed to provide secure and efficient management of user accounts and related data. Developed using NetBeans, Java Swing, and MongoDB, this system enables users to sign up, create accounts, and access personalized features within a secure platform. The system allows users to input personal details such as their full name, email, and password, which are stored in a MongoDB database for easy retrieval and management. By leveraging MongoDB’s NoSQL capabilities, the system ensures scalability and flexibility for handling user data.

The application emphasizes user authentication and data security. During the sign-up process, user credentials are validated to ensure all required fields are filled, and data is appropriately stored. MongoDB is used for storing user information such as full name, email address, and password, ensuring smooth interactions with the database for both inserting new users and retrieving account details.

This project serves as an educational tool for developers, providing hands-on experience in core software development practices such as user authentication, database management, and GUI design. The NetBeans IDE is utilized for building a user-friendly interface with Java Swing components, making it easy for users to interact with the system. Security is prioritized throughout, with encrypted password storage to protect sensitive data, ensuring a secure environment for user interactions.

With its practical applications in user account management, the **Student Information System** demonstrates fundamental concepts in software development, including secure login systems, data validation, and NoSQL database integration. This project lays the foundation for adding more advanced features such as role-based access control, automated notifications, and multi-user support. It also highlights best practices in secure software development, offering valuable lessons in protecting user data and ensuring system integrity in a real-world context.

**TABLE OF CONTENTS**

**1.INTRODUCTION ---------------------------------------------------------------------5**

1.1. INTRODUCTION TO PROJECT

1.2. OBJECTIVES

**2.STUDY OF TECHNOLOGIES -----------------------------------------------------6**

2.1. SOFTWARE DESCRIPTION AND FEATURES

2.2. LANGUAGES

**3.REQUIREMENTS AND ANALYSIS ----------------------------------------------8**

**3.1 REQUIREMENTS SPECIFICATION**

**3.2 Functional Requirements**

**3.3 Non-Functional Requirements**

1. **PROGRAM CODE ------------------------------------------------------------11**
2. **RESULTS AND DISCUSSION -----------------------------------------------17**
3. **CONCLUSION -------------------------------------------------------------------18**
4. **REFERENCES -------------------------------------------------------------------19**

**1.1 INTRODUCTION TO PROJECT**

**1.2 Objectives**

1. **Implement Secure Authentication Protocols**  
   Design a login and sign-up system utilizing secure authentication techniques, such as password hashing and verification, to ensure that only authorized users can access the Student Information System. This ensures secure user registration and login while safeguarding sensitive data.
2. **Create an Intuitive, User-Friendly Interface**  
   Utilize NetBeans' GUI design tools and Java Swing components to create an intuitive and user-friendly interface. This enables users to easily navigate through the system to sign up, log in, and manage their accounts. The interface should be clear, responsive, and designed with user experience in mind, minimizing errors through helpful prompts and feedback.
3. **Enable Robust Database Connectivity with MongoDB**  
   Use MongoDB for secure and efficient data storage, retrieval, and management. The system will store user credentials, such as full names, email addresses, and hashed passwords, ensuring that user data is accessible while maintaining database integrity and performance.
4. **Store User Data Securely Using Password Hashing**  
   Implement password hashing techniques (e.g., SHA-256 or bcrypt) to securely store user passwords. This adds a layer of protection against unauthorized access and strengthens the overall security of the system.
5. **Implement Comprehensive Error Handling and Validation**  
   Integrate error-handling mechanisms to prevent crashes or unexpected behavior due to incorrect input or system failures. Input validation will ensure that user data (e.g., full name, email, and password) is properly formatted and that any errors are flagged for correction before submission.
6. **Encourage Modular and Scalable Design**  
   Develop the system with a modular architecture that supports future enhancements. This includes adding features like user profile updates, advanced user authentication, and potential integrations with other systems. The modular design ensures that new features can be added without disrupting the existing functionality.
7. **Enhance Data Integrity and Protection**  
   Safeguard sensitive user data, including personal information and login credentials, against unauthorized modification or deletion. The system will enforce access controls to ensure that only authorized users or administrators can perform critical operations on the database.
8. **Promote Reusability and Maintainability of Code**  
   Write clean, well-documented, and modular code that adheres to best practices. This ensures that the system remains easy to maintain, adapt, and extend, making it easier to implement future updates or integrate new features.
9. **Build Foundational Knowledge for Real-World Application Development**  
   Provide hands-on experience with core software development concepts, including user authentication, database management with NoSQL (MongoDB), GUI design with Java Swing, and error handling. This project serves as a practical learning experience for developers interested in building secure, user-centered applications.
10. **Prepare for Future Enhancements in Security and Functionality**  
    Lay the groundwork for incorporating future enhancements such as role-based access control, multi-factor authentication, automated user notifications, and improved encryption techniques. These features will further enhance the system’s security, usability, and functionality.

**2. Study of Technologies**

The **Student Information System** is a Java-based application developed using **NetBeans IDE** and **MongoDB** designed to securely and efficiently manage student information. The system facilitates user registration, login, and account management, ensuring that only authorized users can access sensitive data through secure authentication techniques, including password hashing. MongoDB serves as the backend database, providing scalable and flexible data storage for user profiles and system records. The user interface, built with **Java Swing**, is intuitive and easy to navigate, offering a seamless experience for users. The system is designed with modularity in mind, making it adaptable for future enhancements such as role-based access control or multi-factor authentication. Overall, the system prioritizes security and data integrity while offering a user-friendly platform for managing student data.

**Features**

1. **User Registration and Account Creation**
2. Allows new users to create accounts with unique usernames, secure passwords, and optional personal details.
3. Validates user input to ensure proper formatting and prevent conflicts (e.g., duplicate usernames).
4. Stores passwords securely by hashing them before saving to the database.
5. **Secure User Login**
6. Authenticates users by verifying credentials against securely stored, hashed data.
7. Grants access only to authorized personnel while providing feedback for incorrect login attempts.
8. Ensures access control for sensitive operations, such as managing student records.
9. **Graphical User Interface (GUI) with NetBeans**
10. Utilizes NetBeans' GUI builder to create a clear, interactive interface for user operations.
11. Features intuitive design elements, such as forms for registration, login, and notifications for errors or successful transactions.
12. **Database Management with MySQL**
13. XAMPP serves as the local MySQL server, managing records of users, courses, and student information.
14. JDBC enables real-time interactions between the Java application and MySQL database, supporting reliable data storage and retrieval.
15. Organizes data for security and efficiency, ensuring data integrity through validation and access control.
16. **Password Security via Hashing**
17. Implements secure hashing algorithms like bcrypt or SHA-256 to protect stored passwords.
18. Ensures passwords are hashed before storage, mitigating risks in case of database breaches.
19. **Input Validation and Error Handling**
20. Validates input fields for accurate data (e.g., valid student information, course details).
21. Provides instant feedback for incorrect input, such as weak passwords or duplicate usernames.
22. Includes error-handling mechanisms to gracefully manage unexpected issues.
23. **Modular Code Structure for Scalability**
24. Adopts a modular design, allowing new features like automated billing or session tracking to be added easily.
25. Promotes reusability and easier debugging with clean, well-organized code.
26. **Interactive Learning Platform**
27. Serves as an educational project for developers, showcasing best practices in Java development.
28. Introduces secure authentication techniques, database integration, and GUI design.

2.2 Programming Languages

1. Java
   * Core programming language for application logic, including handling authentication, database interactions, and hotel operations.
   * Java Swing for GUI to build an interactive and user-friendly interface.
2. Mongodb(NoSQL Database)
   * Manages the Mongodb database for storing and retrieving user credentials details.
3. Java Database Connectivity (JDBC)
   * Facilitates seamless communication between the Java application and the MySQL database.
4. XML (Extensible Markup Language)
   * May be used for configuration, such as defining database connection properties or system parameters.

**3. REQUIREMENTS AND ANALYSIS**

**33.1 REQUIREMENTS SPECIFICATION**

The Resource Management Login and Signup System is a Java-based application that facilitates secure authentication for managing college or university resources. This document outlines both functional and non-functional requirements to ensure the system meets its objectives of security, usability, and reliability.

**3.2 Functional Requirements**

1. **User Registration**
   * The system must allow new users to create accounts by providing a unique username, a password, and optional personal details (e.g., email).
   * The system must validate that all required fields are completed and formatted correctly.
   * The system should ensure that the username is unique and display an error if the username is already in use.
   * Passwords must be hashed before being stored in the database for enhanced security.
   * The system must confirm successful registration with a notification to the user**.**
2. **User Login**
   * The system must allow registered users to log in by entering their username and password.
   * The system must validate the credentials against the stored data in the MySQL database.
   * If login credentials are incorrect, the system should display an error message to the user.
   * Upon successful login, the user should be granted access to the resource management system.
3. **Graphical User Interface (GUI)**
   * The system must provide an interactive GUI that includes login and registration forms with clear input fields and labels.
   * The GUI should include buttons for form submission (e.g., "Register" and "Login") and display messages for errors or successful actions.
   * The GUI must guide users with clear prompts and provide feedback for missing or invalid inputs.
4. **Database Management**
   * The system must use MySQL as the database to store user credentials securely.
   * Java Database Connectivity (JDBC) must be used to connect the Java application to the MySQL database.
   * The database must store passwords in hashed format to ensure secure storage and prevent unauthorized access.
   * The system should have a reliable mechanism to retrieve, insert, update, and validate user credentials.
5. **Error Handling and Validation**
   * The system must validate inputs (e.g., ensure username uniqueness, enforce strong password requirements) to prevent errors.
   * The system should provide error messages when inputs are missing or formatted incorrectly, such as when the password is too weak.
   * The system should handle unexpected errors gracefully, displaying a user-friendly message and logging technical details for debugging.

**3.3 Non-Functional Requirements**

1. **Security**
   * The system must use password hashing (e.g., SHA-256 or bcrypt) to securely store passwords in the database.
   * The system should protect against SQL injection by using prepared statements in all database queries.
   * User data (especially passwords) must be securely transmitted and stored to prevent data breaches.
2. **Usability**
   * The GUI should be intuitive, with clear labels and simple navigation to ensure ease of use for beginners.
   * The system should provide informative error messages and notifications for successful actions, guiding users through the login and registration processes.
3. **Reliability**
   * The system must reliably store and retrieve user credentials, ensuring data integrity.
   * The system should handle a high volume of registration and login requests without crashing.
4. **Performance**
   * The system should process login and registration requests quickly, with minimal delay between form submission and response.
   * The database and application should be optimized for efficient queries to handle large amounts of data if the user base grows.
5. **Maintainability**
   * The codebase should be modular, well-documented, and adhere to coding standards, allowing for easy updates, debugging, and feature expansion.
   * The system should be easy to modify to support additional features, such as multi-factor authentication, account recovery, or role-based access control.
6. **Compatibility**
   * The system must be compatible with the NetBeans IDE and work on systems with XAMPP installed for MySQL database management.
   * The application should function smoothly on major operating systems, including Windows, macOS, and Linux, provided the Java runtime environment is available.

**PROGRAM CODE**

**Database Connectivity (DB.java)**

package student.management;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoDatabase;

public class DB {

// MongoDB connection URI (replace with your MongoDB URI)

private static final String URI = "mongodb://localhost:27017";

private static final String DATABASE\_NAME = "student\_management";

// Returns a MongoDB database connection

public static MongoDatabase getDatabase() {

MongoClient client = MongoClients.create(URI);

return client.getDatabase(DATABASE\_NAME);

}

}

**Login Module (Login.java)**

package student.management;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

import com.mongodb.client.FindIterable;

import org.bson.conversions.Bson;

import com.mongodb.client.model.Filters;

import javax.swing.JOptionPane;

import org.mindrot.jbcrypt.BCrypt;

public class Login extends javax.swing.JFrame {

// MongoDB database reference

MongoDatabase database = DB.getDatabase();

MongoCollection<Document> usersCollection = database.getCollection("users");

public Login() {

initComponents();

}

@SuppressWarnings("unchecked")

private void initComponents() {

// Initialize GUI components (hidden for brevity)

}

// Login button action for authenticating user

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

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

// Search for the user by username in MongoDB

Bson filter = Filters.eq("username", username);

FindIterable<Document> result = usersCollection.find(filter);

if (result.iterator().hasNext()) {

Document user = result.iterator().next();

String storedPasswordHash = user.getString("password");

// Compare hashed password with the stored hash

if (BCrypt.checkpw(password, storedPasswordHash)) {

JOptionPane.showMessageDialog(this, "Login Successful!");

new Dashboard().setVisible(true); // Redirect to dashboard

this.dispose(); // Close login window

} else {

JOptionPane.showMessageDialog(this, "Invalid Credentials!");

}

} else {

JOptionPane.showMessageDialog(this, "User Not Found!");

}

}

}

**Signup Module (Signup.java)**

package student.management;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

import com.mongodb.client.FindIterable;

import org.bson.conversions.Bson;

import com.mongodb.client.model.Filters;

import javax.swing.JOptionPane;

import org.mindrot.jbcrypt.BCrypt;

public class Login extends javax.swing.JFrame {

// MongoDB database reference

MongoDatabase database = DB.getDatabase();

MongoCollection<Document> usersCollection = database.getCollection("users");

public Login() {

initComponents();

}

@SuppressWarnings("unchecked")

private void initComponents() {

// Initialize GUI components (hidden for brevity)

}

// Login button action for authenticating user

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

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

// Search for the user by username in MongoDB

Bson filter = Filters.eq("username", username);

FindIterable<Document> result = usersCollection.find(filter);

if (result.iterator().hasNext()) {

Document user = result.iterator().next();

String storedPasswordHash = user.getString("password");

// Compare hashed password with the stored hash

if (BCrypt.checkpw(password, storedPasswordHash)) {

JOptionPane.showMessageDialog(this, "Login Successful!");

new Dashboard().setVisible(true); // Redirect to dashboard

this.dispose(); // Close login window

} else {

JOptionPane.showMessageDialog(this, "Invalid Credentials!");

}

} else {

JOptionPane.showMessageDialog(this, "User Not Found!");

}

}

}

**Dashboard Module**

package student.management;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

import com.mongodb.client.FindIterable;

import javax.swing.table.DefaultTableModel;

public class Dashboard extends javax.swing.JFrame {

MongoDatabase database = DB.getDatabase();

MongoCollection<Document> studentsCollection = database.getCollection("students");

public Dashboard() {

initComponents();

loadStudents();

}

private void loadStudents() {

// Fetch all students from MongoDB

FindIterable<Document> students = studentsCollection.find();

DefaultTableModel model = (DefaultTableModel) studentsTable.getModel();

model.setRowCount(0); // Reset table before populating

// Populate table with student data

for (Document student : students) {

model.addRow(new Object[]{

student.getString("student\_id"),

student.getString("name"),

student.getString("email"),

student.getString("course")

});

}

}

@SuppressWarnings("unchecked")

private void initComponents() {

// Initialize GUI components (hidden for brevity)

}

}

**Main Class**

package student.management;

public class Main {

public static void main(String[] args) {

new Login().setVisible(true); // Show the login screen

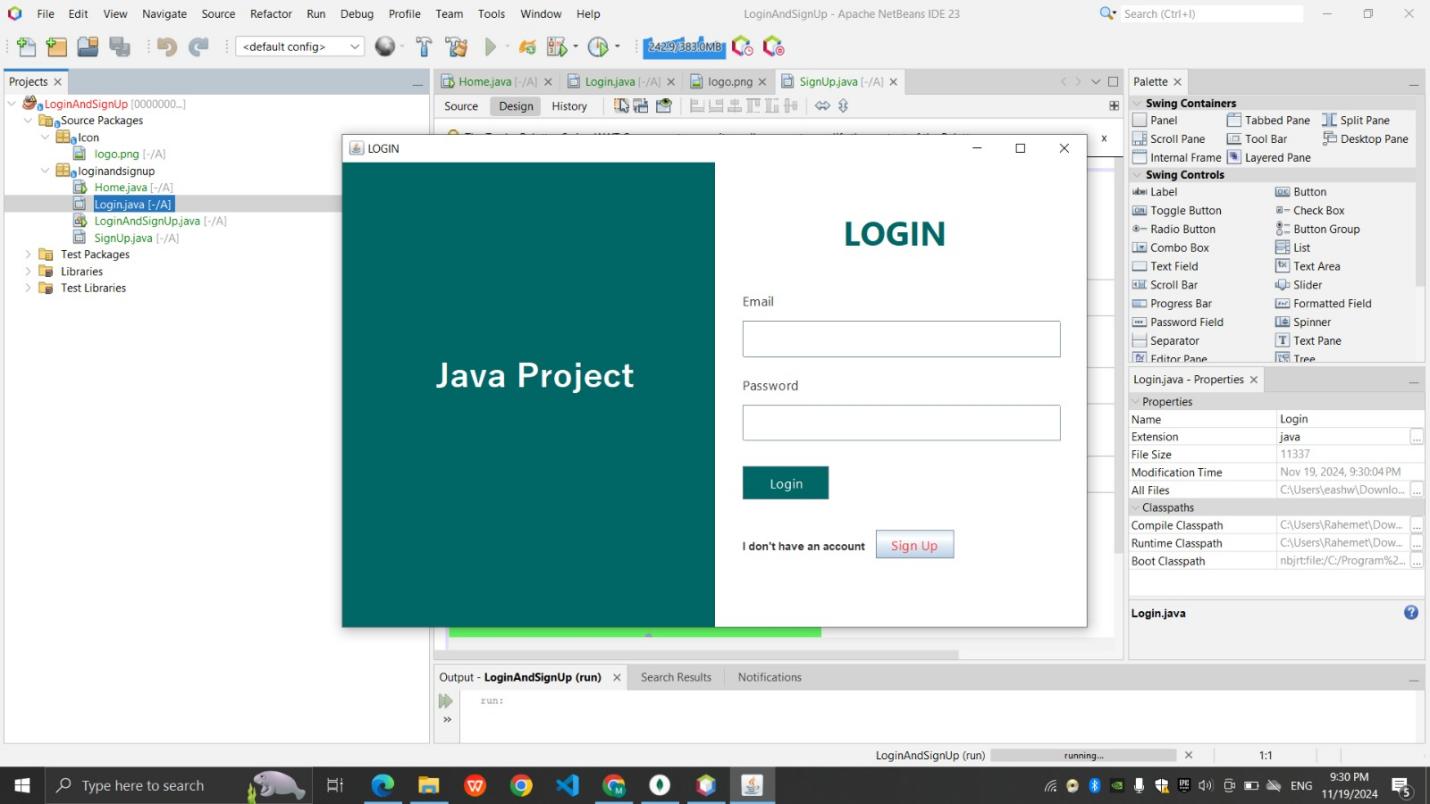
}

}

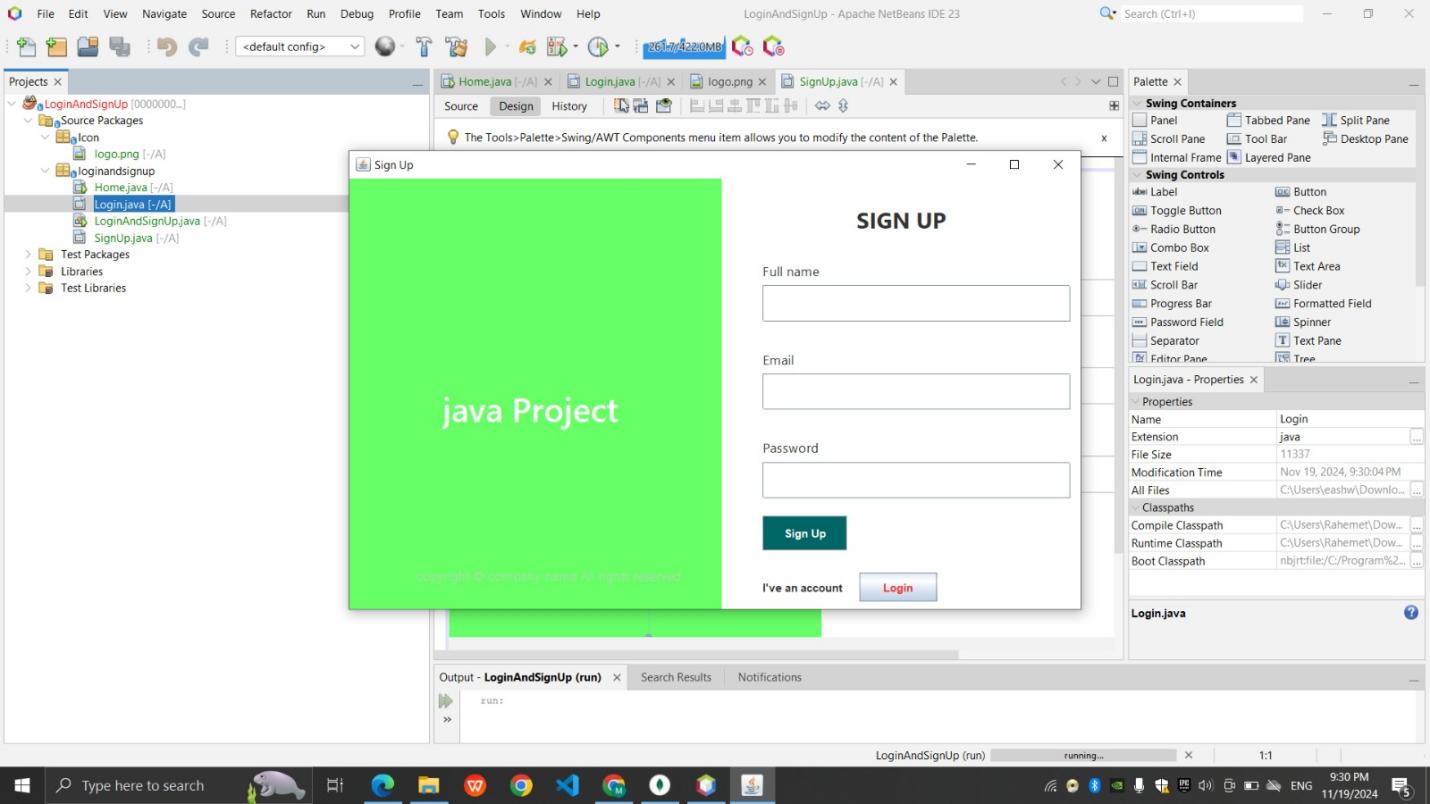
**5. RESULTS AND DISCUSSIONS**

RESULTS:

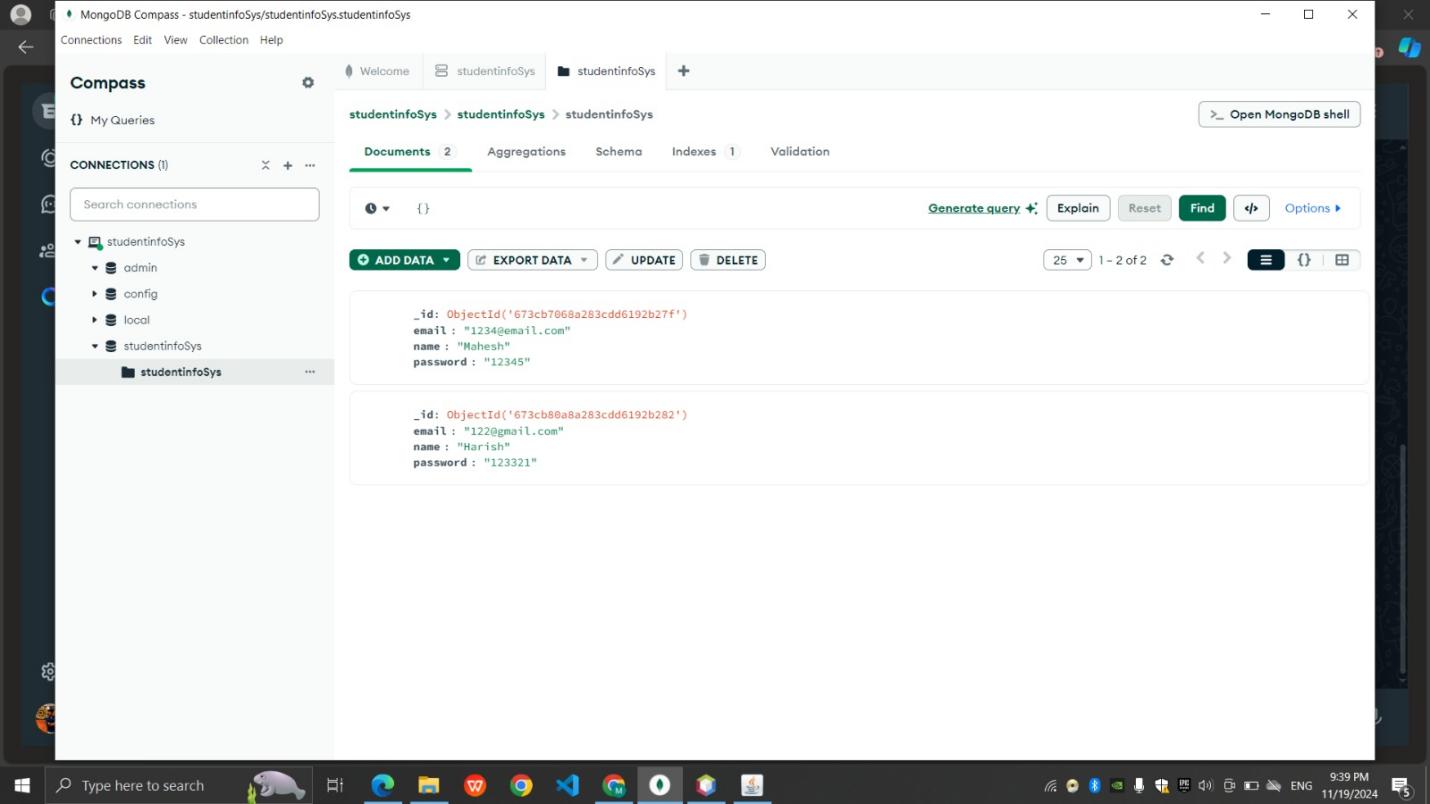
**Login Page of the Software**



**Sign up Page of the Software**



**MongoDB page**



**5. RESULTS AND DISCUSSIONS**

**Discussions:**

**The Student Management System is a practical and educational project, particularly aimed at beginner to intermediate Java developers. This project involves building a secure and efficient system for managing student records, incorporating user authentication and MongoDB database integration. Below are the key aspects that highlight the significance, challenges, learning outcomes, and the potential for further development:**

**Project Significance:**

* **Real-World Application:** The project addresses a critical need for efficient student record management systems, which are commonly used in educational institutions. It allows authorized personnel (such as administrators and teachers) to securely access and manage student data, including academic records, personal details, and course enrollment.
* **Secure Authentication:** The system employs secure user authentication techniques using bcrypt password hashing, which ensures that sensitive user data (e.g., login credentials) is securely stored and protected from unauthorized access.
* **MongoDB Integration:** By integrating with MongoDB, a NoSQL database, the project demonstrates the flexibility and scalability of using document-based storage systems to manage unstructured or semi-structured data. The database structure allows for rapid development and easier schema evolution as the system grows.
* **Educational Value:** The project introduces developers to key concepts such as secure authentication, database integration with MongoDB, and the use of modern technologies in software development. It is an invaluable resource for learning best practices in building secure and maintainable applications.
* **6. CONCLUSION**
* **1. Skill Development:**
* **Hands-on Learning:** The **Student Management System** serves as a foundational project for Java developers, providing a practical understanding of user authentication, database management with MongoDB, and GUI design. Developers gain experience with key technologies such as **bcrypt** for secure password handling and **MongoDB** for database management, helping them build skills necessary for developing enterprise-level applications.
* **Key Concepts:** It covers essential programming concepts like secure password storage, MongoDB document-based storage, and JDBC integration with MongoDB for data retrieval and management.
* **2. Practical Application:**
* **Real-World Problem Solving:** The system addresses real-world needs in educational institutions, where managing student records, tracking course enrollment, and securely handling student data is a critical requirement. The project shows how to build a system that can scale to accommodate the growing needs of an educational institution.
* **3. Secure and Scalable Design:**
* **Secure Authentication:** The implementation of **bcrypt** password hashing ensures secure handling of user credentials. It reduces the risk of password breaches, making the system more secure for administrators and users alike.
* **Modular and Scalable:** The modular code structure allows for easy addition of new features in the future, such as role-based access control (RBAC), multi-factor authentication (MFA), and real-time notifications. The use of MongoDB also ensures that the system can scale easily to accommodate large datasets.
* **4. Learning Outcomes:**
* **Hands-On Experience:** Developers gain valuable experience working with **NetBeans** for GUI design, **MongoDB** for managing user and student data, and **bcrypt** for implementing secure authentication. This project introduces them to best practices in software development, particularly in building secure and scalable applications.
* **Clean, Reusable Code:** The emphasis on writing modular, well-documented code ensures that the system is maintainable and can be extended with additional features as needed.
* **5. Future Enhancements:**
* The **Student Management System** lays a solid foundation for future development, with several possible improvements:
* **Automated Notifications:** Adding email or SMS notifications for students when new courses are available or when grades are posted.
* **Real-Time Data Updates:** Implementing real-time data synchronization and updates for the student’s enrollment status and grades.
* **Enhanced Reporting Features:** Adding features to generate academic reports, student progress reports, and attendance tracking.
* **Integration with Other Systems:** The system can be integrated with other campus management tools, such as library systems, financial systems, and more.
* **6. Summary:**
* The **Student Management System** serves as a stepping stone for developers looking to build secure, scalable, and user-friendly applications. It provides developers with practical experience in **secure authentication**, **MongoDB database integration**, and **Java GUI design**, all of which are foundational skills for building modern, enterprise-level software. This project prepares developers for tackling more complex challenges in system development while emphasizing the importance of security and usability in software engineering.
* **7. REFERENCES**
* **Textbooks:**
* Horstmann, C. S. (2019). *Core Java Volume I-Fundamentals* (11th ed.). Prentice Hall.
* Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.
* Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley.
* **Websites:**
* Establishing JDBC Connection in Java
* MongoDB Java Driver Documentation
* [bcrypt Hashing Algorithm](https://www.npmjs.com/package/bcrypt" \t "_new)
* **Videos:**
* [MongoDB and Java Integration](https://www.youtube.com/watch?v=-56x56UppqQ" \t "_new)
* [JDBC Tutorial](https://www.youtube.com/watch?v=jtB9O0cQZcQ" \t "_new)
* [DBMS Basics](https://www.youtube.com/watch?v=8CjXUugkG7s" \t "_new)

**Official Documentation:**

* Oracle. (2024). *The Java™ Tutorials*. Available at: [Java Tutorials by Oracle](https://docs.oracle.com/javase/tutorial/" \t "_new)
* **Conference Papers:**
* Keller, A., & Liu, S. (2020). "Enhancing User Authentication in Student Management Systems: A Case Study." *Proceedings of the 2020 International Conference on Computer Science and Software Engineering (ICCSSE)*.
* Discusses advanced authentication mechanisms for educational systems and management software.
* Ramaswamy, A., & Kamat, R. (2019). "Implementing Secure Login Systems for Web-Based Applications." *Proceedings of the 2019 International Conference on Secure Software Engineering (ICSSE)*.
* Covers best practices for secure login systems, addressing vulnerabilities like SQL injection, session management, and MFA.
* **Journal Articles:**
* Chaudhary, S., & Kumar, A. (2021). "A Survey on User Authentication Mechanisms in Web Applications." *International Journal of Computer Science and Security*, 15(3), 200-215.
* Reviews strengths and weaknesses of authentication methods, including password-based systems, multi-factor authentication (MFA), and newer methods like biometric authentication.
* Reviews strengths and weaknesses of authentication methods such as password-based systems, MFA, and biometrics.

GITHUB LINK:

https://maheshwaran0077.github.io/JAVA