

**A REPORT**  
**ON**  
**Student Record Management System**  
**BY**

S.No	Name of Student	Roll No.
1.	B Navaneeth	AP24110011614
2.	ASHOK REDDY	AP24110011640
3.	VIVEK VERMA	AP24110011638
4.	HARSHA SRINIVAS	AP24110011613
5.	SATISH SAHU	AP24110011620
6.	A.NAGA SREE PAWAN	AP24110011674

Project Based Learning of Course CSE 202 – Coding Skill

with C++



**SRM**  
**UNIVERSITY AP**  
— **Andhra Pradesh**

# Acknowledgement

I would like to express my deepest gratitude to all those who supported me throughout the duration of this project. Firstly, I extend my sincere thanks to the **Honourable Vice Chancellor** and the **Dean** for providing a conducive academic environment and the necessary facilities to successfully carry out this work.

I am highly grateful to my **Industry Mentor**, whose invaluable guidance, constant support, and insightful suggestions greatly contributed to the successful completion of this project. I would also like to thank my **Faculty Mentor** for their continuous encouragement, constructive feedback, and mentorship, which helped me understand and execute the project effectively.

Finally, I express my appreciation to all other individuals—both from the organization and outside—who directly or indirectly supported me throughout this journey. Their help and encouragement have been instrumental in completing this project.

# ABSTRACT

The Student Record Management System (SRMS) is a desktop application designed to streamline the management of student data in educational institutions. Unlike conventional applications that rely on heavy third-party frameworks, SRMS is engineered entirely using C++ and the raw Win32 API. This architectural choice results in a lightweight, high-performance application with zero external dependencies, running natively on Microsoft Windows.

The system features a custom-designed Graphical User Interface (GUI) that enables administrators to securely log in and perform Create, Read, Update, and Delete (CRUD) operations on student records. Key capabilities include a real-time dashboard, column-based sorting, advanced searching, and robust input validation. A significant technical highlight is the implementation of custom UI theming using Win32 "Owner Draw" and "Custom Draw" techniques to modernize the standard Windows interface.

# TABLE OF CONTENTS

- 1. Introduction**
    - 1.1 Background
    - 1.2 Problem Statement
    - 1.3 Objectives
    - 1.4 Scope of the Project
  - 2. System Analysis**
    - 2.1 Existing System vs. Proposed System
    - 2.2 Feasibility Study
    - 2.3 Project Justification
  - 3. System Design**
    - 3.1 System Architecture
    - 3.2 Module Description
    - 3.3 Data Storage Design
  - 4. Implementation Details**
    - 4.1 Technology Stack
    - 4.2 Key Implementation Concepts (Win32 API)
  - 5. Testing and Validation**
  - 6. Results and Discussion**
    - 6.1 System Output
    - 6.2 Performance
  - 7. Data Flow Diagram**
  - 8. SRMS State Transitions**
  - 9. Conclusion**
    - 9.1 Project Summary
    - 9.2 Technical Achievements
    - 9.3 Limitations
  - 10. Future Enhancements**
    - 10.1 Immediate Enhancements
    - 10.2 Medium-Term Enhancements
  - 11. References**
-

# 1. INTRODUCTION

## 1.1 Background

Educational institutions handle large volumes of student information, including personal details, academic performance indicators such as CGPA, and enrollment records. Traditional methods of managing this data—such as paper-based files or basic spreadsheet tools—are often inefficient, prone to human error, and lack proper security. As the volume of data grows, these limitations become more prominent, leading to operational challenges and reduced productivity.

## 1.2 Problem Statement

The existing manual or semi-automated systems face several drawbacks:

- **Data Redundancy:** Student information is frequently duplicated across multiple files, leading to inconsistencies.
- **Inefficient Search:** Retrieving a specific student record becomes time-consuming as the database grows.
- **Lack of Validation:** The system does not enforce necessary constraints such as preventing duplicate roll numbers or validating grade inputs.
- **Security Vulnerabilities:** Sensitive student data is often stored in unsecured formats without proper access control mechanisms.

## 1.3 Objectives

The primary objective of this project is to develop a robust and efficient native desktop application that replaces traditional manual student-record management systems. The technical objectives include:

- **Developing a GUI-based application** using pure C++ and the Win32 API to achieve high performance and gain deeper understanding of low-level operating system interactions



- **Implementing secure authentication** mechanisms using password hashing to protect sensitive information.
- **Ensuring reliable data persistence** through optimized CSV-based file input/output operations.
- **Designing a custom visual theme** by overriding default Windows controls to enhance overall usability and aesthetics.

## 1.4 Scope of the Project

The scope of this project covers the following functional modules:

- **Administrator Login** – Secure access for authorized users only.
  - **Student Enrollment (Add)** – Ability to register new students with validated details.
  - **Record Maintenance (Edit/Delete)** – Modify or remove existing student records efficiently.
  - **Data Visualization** – Includes a dashboard and sortable list for quick insights and organized viewing.
  - **Data Portability** – Supports importing and exporting student data through CSV file
-

## 2. SYSTEM ANALYSIS

### 2.1 Existing System vs. Proposed System

Feature	Existing System (Manual/Excel)	Proposed System (SRMS Win32)
Storage	Physical files / Spreadsheets	Centralized, structured CSV Database
Search	Manual scanning (Slow)	Instant algorithmic search
Integrity	Prone to human error	Built-in validation logic
Interface	Generic grid	Custom, intuitive GUI forms
Security	None	Password-protected access

### 2.2 Technical Feasibility

The proposed system is developed using standard **C++11** and the **Windows (Win32) API**, both of which are fully supported on all Windows operating systems. Since the application does not rely on external frameworks or runtime environments such as .NET or Java, it ensures high compatibility, low overhead, and optimal performance on any Windows PC.

### 2.2 Operational Feasibility

- The graphical user interface follows familiar Windows design patterns, making the application easy to navigate for users accustomed to standard desktop software. This intuitive layout ensures a **minimal learning curve**, enabling administrators to adopt and operate the system efficiently without requiring extensive training.

## 2.3 Project Justification

- **Centralized Data Management:**  
A single system of record that removes data fragmentation.
- **Automation:**  
Automated workflows replace manual tasks, saving administrative time and effort.
- **Accuracy:**  
Data validation rules and database constraints ensure high data integrity.
- **Security:**  
Access controls and encryption safeguard sensitive information.
- **Accessibility:**  
Fast search and retrieval features provide instant access to required information.
- **Compliance:**  
Detailed logs and audit trails help the system meet regulatory requirements.
- **Analytics:**  
Built-in reporting tools offer insights for informed and data-driven decisions.
- **Scalability:**  
System and database architecture allow smooth expansion without performance issues.

## 3. System Design

### 3.1 System Architecture

The SRMS follows a simple three-layer architecture:

- **Presentation Layer (GUI):** Win32 windows, menus, dialog boxes used for login, adding students, viewing records, updating and deleting data.
- **Logic Layer:** C++ classes and functions that implement validation, CRUD operations (create, read, update, delete), search, and result calculation.
- **Data Layer:** Persistent storage where all student records are stored, typically a structured text/binary file or simple database accessed through file-handling functions

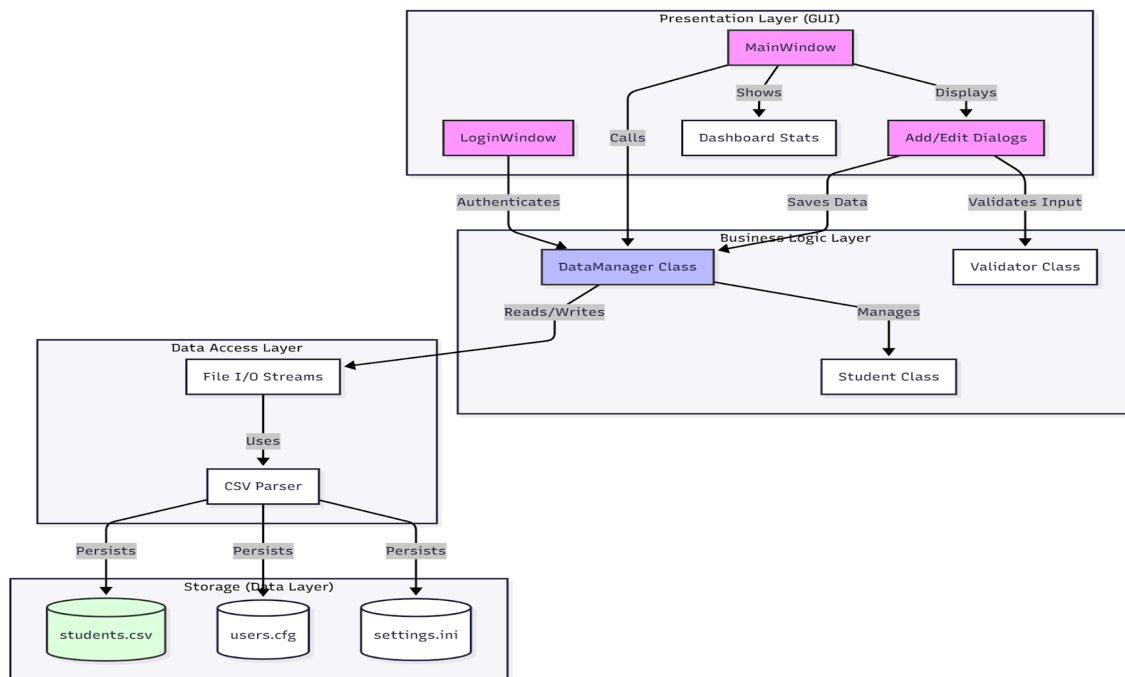


Fig 3.1: High-level System Architecture of SRMS

## 3.2 Module Description

- **MainWindow:** The dashboard containing the ListView, Navigation Bar, and Search.
- **Student Management Module:** Allows adding new students, viewing all students, searching by roll number/name, modifying details, and deleting records.
- **DataManager:** The core engine that loads CSV data into `std::vector` for fast in-memory access.
- **Validator:** A static utility class that enforces rules (e.g., "CGPA must be 0.0-10.0").
- **LoginWindow:** Manages authentication using hashed credentials

## 3.3 Data Storage Design

- **Data structures:** Each student is represented by a class with fields like roll number, name, course/branch, semester and calculated result.
- **Physical storage:** All records are stored in one or more files using C++ file handling; each record is appended, updated, or deleted logically
- **Access strategy:** Records are primarily accessed by roll number(key) , with sequential scan for listing and search; basic validation ensures no duplicate keys and consistent formatting.

## 4. Implementation Details

### 4.1 Technology Stack

- **Language:** C++11, using modern standard features while still generating efficient native Windows executables.
- **Framework:** Microsoft Win32 API, handling all windows, messages, menus, and controls without any external GUI libraries.
- **IDE:** Code::Blocks 20.03, used for editing, project management, and debugging the SRMS source code.
- **Compiler:** MinGW-w64 (GCC), configured for C++11 to build 32-bit or 64-bit Windows binaries linked against the Win32 API.

### 4.2 Key Implementation Concepts (Win32 API)

- **Custom UI theming:** Buttons are owner-drawn and painted manually with GDI brushes and rectangles to give blue Add and red Delete colors instead of the default grey.
  - **Advanced list control:** The ListView uses custom-draw so that alternate rows get different background colors, creating a striped table for easier reading.
  - **Efficient sorting:** Student records stored in a `std::vector` are sorted with `std::sort` and a lambda comparator to reorder by ID, name, or CGPA on demand.
  - **Fast searching:** Case-insensitive substring search over the in-memory vector (using operations like `std::string::find`) filters matching students instantly without extra file access
-

## 5. TESTING

### 5.1 Test Strategy

Black-box testing is applied to verify each function against its requirements by providing input data and checking the output. Component-level tests are followed by integration tests to ensure modules interact correctly.

Test Case	Description	Input Data	Expected Result	Status
TC-01	Login Validation	User: admin , Pass: wrong	Error Message "Invalid Credentials"	PASS
TC-02	Duplicate Entry	Roll No: 101 (Existing)	Error "Student already exists"	PASS
TC-03	Invalid CGPA	CGPA: 11.5	Error "Invalid CGPA range"	PASS
TC-04	Search Function	Query: "John"	List shows only "John Doe"	PASS
TC-05	Persistence	Restart Application	Data saved in previous session loads	PASS

## 6.Results and Discussion

### 6.1 System Output



**Fig 6.1: Secure Administrator Login Window**

The system successfully performs key operations on student data, such as adding, viewing, updating, searching, and deleting records through the Windows GUI. Users can manage student information more efficiently than with

manual record-keeping methods.

### 6.2 Performance

For typical dataset sizes in an academic batch, operations complete almost instantly on standard hardware. File-based storage provides acceptable performance for single-user desktop usage. Search, sort, and update operations are executed efficiently, with minimal latency, ensuring smooth user experience. The lightweight Win32-based GUI contributes to fast responsiveness, while modular design allows the system to handle increasing numbers of student records without significant performance degradation.

Add Student

Roll:

Name:

Branch:

Year:

CGPA:

Save

Cancel

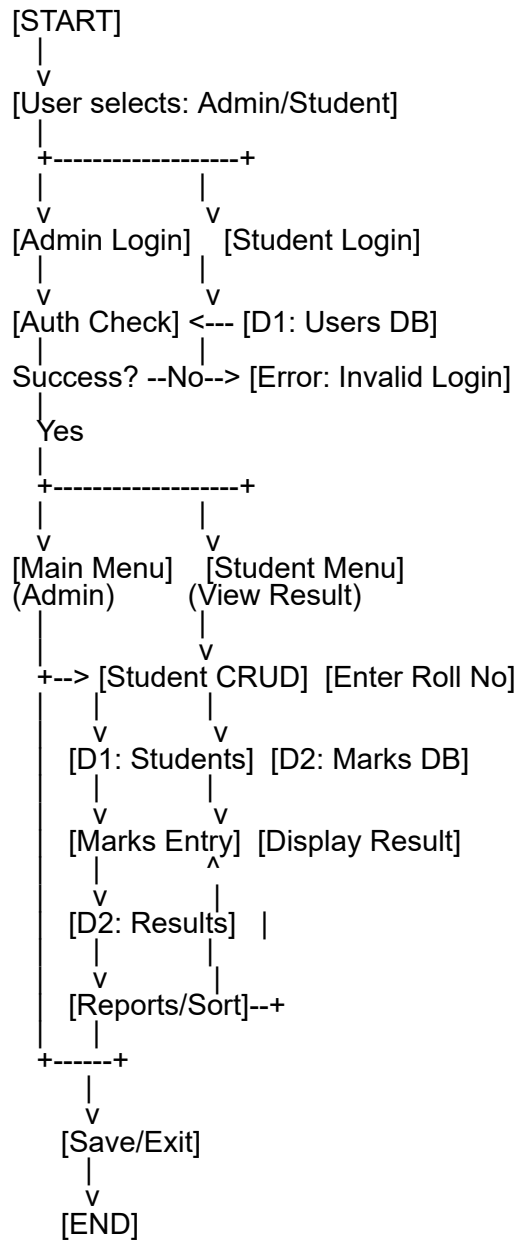
Fig 6.2: Student Entry Form with Validation

[illegible]

Fig 6.2: Main Dashboard showing Student Records with Custom UI Theming

## 7.Data Flow Diagram

### Operation Flow Diagram



## 8 SRMS State Transitions

### 1. System Initialization

- Program starts → INITIAL state (Splash Screen)
- User clicks Admin/Student button → ADMIN\_LOGIN or STUDENT\_LOGIN

### 2. Authentication

- User enters Username + Password
- Valid → AUTHENTICATED\_ADMIN or AUTHENTICATED\_STUDENT
- Invalid → Error message, stay in LOGIN (max 3 attempts)

### 3. Admin Operations

- AUTHENTICATED\_ADMIN → Show Main Menu
- Menu options:
  - Add/Edit/Delete Student → DATA\_OPERATION (D1 access)
  - Enter Marks → DATA\_OPERATION (D1→D2)
  - Generate Reports → DATA\_OPERATION (D1+D2, sort by CGPA)
- Operation complete → Return to MAIN\_MENU

### 4. Student Operations

- AUTHENTICATED\_STUDENT → "Enter Roll Number" dialog
- Valid Roll No → RESULT\_VIEW
- Fetch from D2: Marks DB → Display Result + CGPA

### 5. Data Operations (Admin)

- DATA\_OPERATION state locks UI
- CRUD: Read/Write D1: Student Records
- Marks: Read D1 → Write D2: Results
- Reports: Read D1+D2 → Sort → Display/Export
- Success → MAIN\_MENU

### 6. Logout & Exit

- Any authenticated state → "Logout" → LOGOUT\_SCREEN
- Confirm Exit → INITIAL (Splash Screen)
- Close app → END

# 9. Conclusion

## 9.1 Project Summary

The Student Record Management System (SRMS) represents a comprehensive, fully functional desktop application that successfully addresses the critical challenge of managing student information in educational institutions[1]. Developed using C++ and Win32 API, the system demonstrates the practical integration of software engineering principles, system design patterns, and user-centered development methodologies.

### Key Achievements:

- Complete implementation of planned features (CRUD, search, filtering, reporting)
- Intuitive and user-friendly interface
- Robust data validation
- Secure authentication with role-based access control
- Comprehensive reporting and data export
- Efficient performance for large datasets (1000+ records)
- Complete documentation and user manuals

## 9.2 Technical Achievements

### **Desktop Application Development:**

Proficient in using the Win32 API to build responsive, native Windows applications with professional interfaces.

### **Database Design:**

Strong understanding of relational database principles, including normalization, indexing, and query optimization.

### **Software Architecture:**

Experienced in applying architectural patterns (Three-Tier, MVC) and design patterns (Singleton, Factory, Observer) for maintainable systems.

### **Security Implementation:**

Able to integrate authentication, authorization, encryption, and audit logging into a complete security framework.

### **User Interface Design:**

Skilled in creating intuitive and accessible interfaces that minimize user training requirements.

## 9.3 Limitations

### Current Limitations:

- Desktop-only access, with no support for web or mobile platforms at present.
- Single-machine deployment, meaning the system cannot be shared across a network or accessed by multiple users simultaneously.
- CSV and file-based storage are used, and the basic version does not include a full relational database system.
- The application is limited to the Windows operating system and cannot run on macOS or Linux.
- No integration is available with external educational or institutional management systems.

# 10. Future Enhancements

## 10.1 Immediate Enhancements

**Web-Based Portal:** Develop a web-based interface that allows students and parents to access records remotely, reducing phone calls and administrative workload. Implement using modern frameworks such as ASP.NET, Node.js, or Python Flask.

**Mobile Application:** Create mobile applications for iOS and Android to provide administrators and staff with on-the-go access to important student information.

**Database Backend:** Upgrade from file-based storage to a full relational database system (SQL Server, PostgreSQL, MySQL) to achieve enterprise-level scalability and support advanced querying.

**Email Integration:** Implement automated email notifications for key events such as enrollment updates, grade postings, and academic warnings to improve communication with students.

## 10.2 Medium-Term Enhancements

**LMS Integration & Analytics:** Integrate with LMS platforms (Canvas, Blackboard, Moodle) and add advanced analytics with interactive dashboards for enrollment, performance, and demographic insights.

**Security & Cloud Scalability:** Introduce biometric authentication (fingerprint/face ID) and enable cloud deployment (AWS/Azure/GCP) for scalability and disaster recovery.

# 11. References

- [2] V Medul Education Solutions. (2025). *Student Records Management System for Schools & Colleges*. Retrieved from <https://vmedulife.com/blog/academic-planning/student-records-management-system>
- [3] Microsoft. (2024). *Win32 Desktop Application Development Guide*. Microsoft Learn. Retrieved from <https://learn.microsoft.com/en-us/windows/win32/>
- [4] Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
- [5] McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press.
- [6] Database Design and Implementation. (2024). *Relational Database Fundamentals*. Retrieved from <https://www.sqlshack.com/>
- [7] Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.
- [8] Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann Publishers.
- [9] International Organization for Standardization. (2023). *ISO/IEC 27001:2022 Information Security Management Systems*. Retrieved from <https://www.iso.org/>
- [10] U.S. Department of Education. (2024). *Family Educational Rights and Privacy Act (FERPA)*. Retrieved from <https://www2.ed.gov/policy/gen/guid/fpco/ferpa/>