SINDH MADRESSATUL ISLAM UNIVERSITY

STUDENT MANAGEMENT SYSTEM

A Project Report Submitted in Partial Fulfillment of the Requirements for the Course

Data Structures and Algorithms

(BS Computer Science)

Submitted By:

Muhammad Saad (CSC-24F-025)

Submitted To:

Miss Syeda Nazia Ashraf

Department of Computer Science

Sindh Madressatul Islam University, Karachi

December 30, 2025

2. Introduction

2.1 Background

Modern academic institutions require efficient systems to manage student records, registration requests, and performance tracking. While numerous software solutions exist, they often rely on high-level abstractions that obscure fundamental algorithmic operations. This project was conceived to bridge the gap between theoretical data structures knowledge and practical software engineering by implementing a complete management system using only manual, low-level data structure implementations.

2.2 Problem Statement

Traditional academic projects often utilize Java's built-in collections (ArrayList, LinkedList, Queue), which prevents students from understanding underlying algorithmic complexities. The challenge was to develop a fully functional Student Management System that:

- Implements custom dynamic arrays, queues, and searching/sorting algorithms from scratch
- Provides role-based access control (Students vs. Administrators)
- Handles registration workflows with approval queues
- Demonstrates real-time algorithm execution through comprehensive logging
- Maintains data integrity without external libraries or frameworks

2.3 Objectives

1. **Primary Objective:** Design and implement a GUI-based application demonstrating core DSA concepts (arrays, queues, searching, sorting) without using built-in Java collections.

1. **Functional Objectives:**

- Enable student authentication and profile management
- Implement administrator-controlled registration approval workflows

- Provide multiple search algorithms (Linear & Binary) with performance visualization
- Demonstrate three sorting algorithms (Bubble, Selection, Insertion) with step-by-step trace
- Generate statistical reports (average marks, top scorer)

3. Educational Objectives:

- Visualize memory management during dynamic array resizing
- Trace queue operations (enqueue/dequeue) with front/rear pointer movements
- Log every comparison and swap during sorting operations
- Expose index-by-index traversal during searching

2.4 Scope

In-Scope:

- Role-based authentication and authorization
- CRUD operations on student records using custom StudentArray
- Queue-based registration management using RegistrationQueue
- Four searching methods (Linear/Binary by Roll Number and Name)
- Three sorting methods on student attributes
- Report generation and grade calculation
- Graphical User Interface (GUI) using Java Swing for better user interaction
- Extensive logging of DSA operations in console and GUI alerts

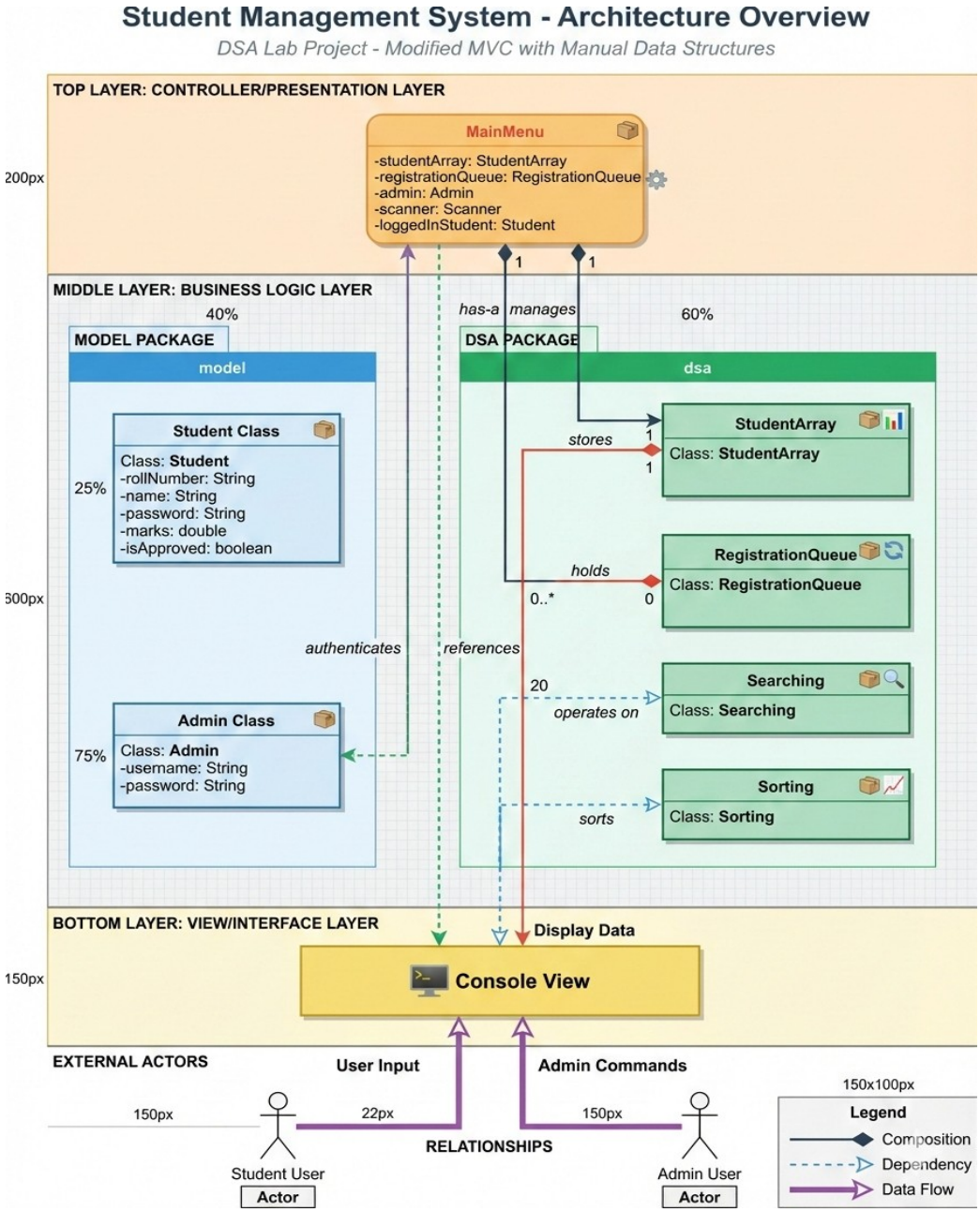
Out-of-Scope:

- Persistent storage (file/database) – data is volatile
- Multi-threading or network capabilities

3. System Design

3.1 Architecture Overview

The system follows a modified MVC (Model-View-Controller) architecture adapted for GUI applications.



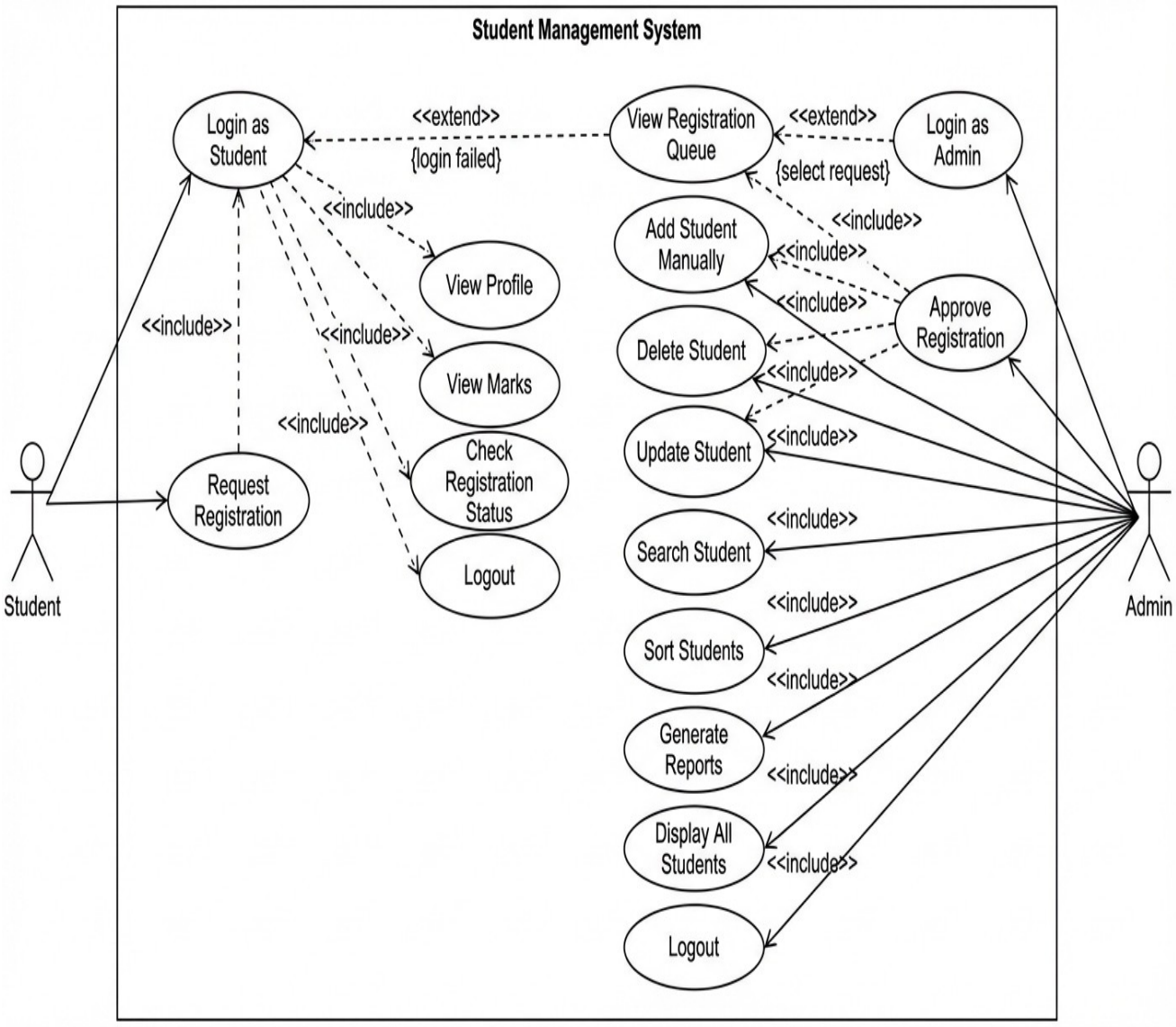
Key Design Principles:

- **Separation of Concerns:** Entity classes (Student, Admin) are independent of data structures
- **Manual Implementation:** All data structures are built from primitive arrays
- **Extensive Logging:** Every DSA operation is prefixed with [DSA] for traceability

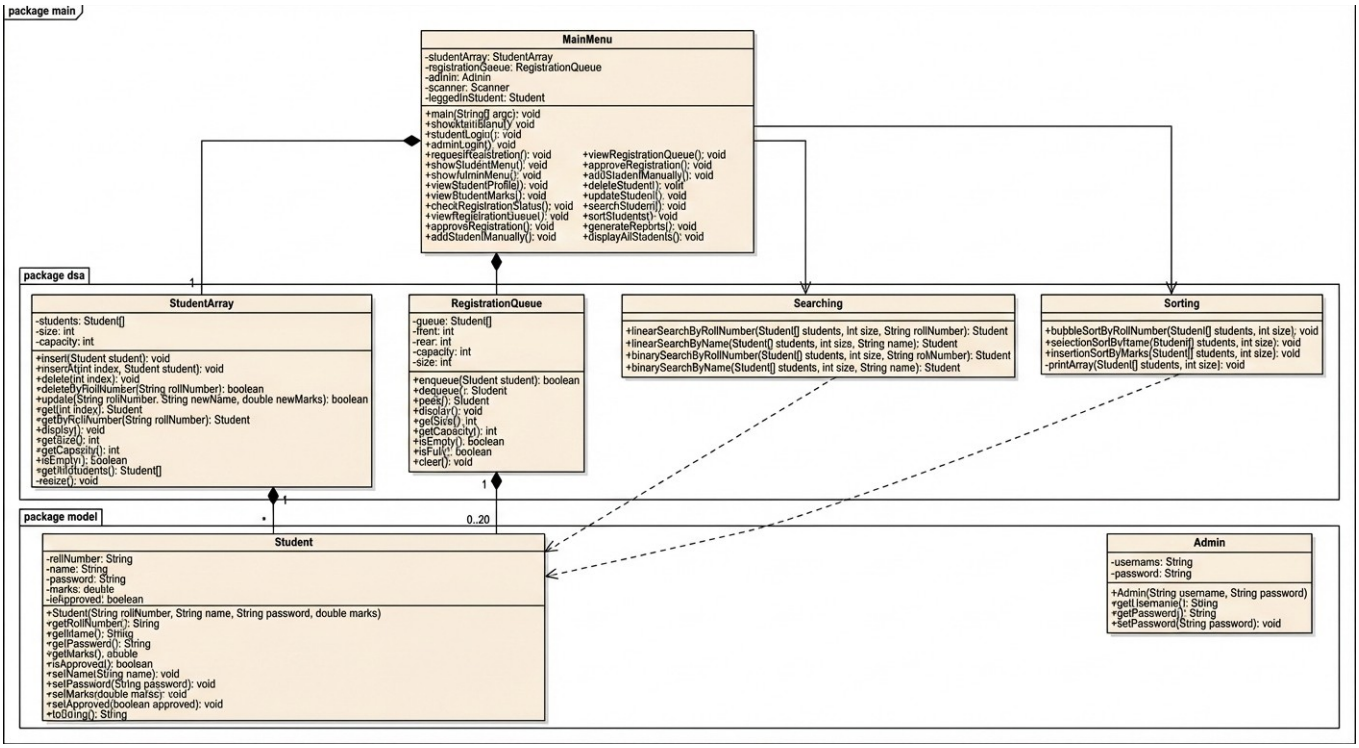
- **GUI-Driven Interface:** Button-based navigation with panels and frames for clear state management

3.2 UML Diagrams

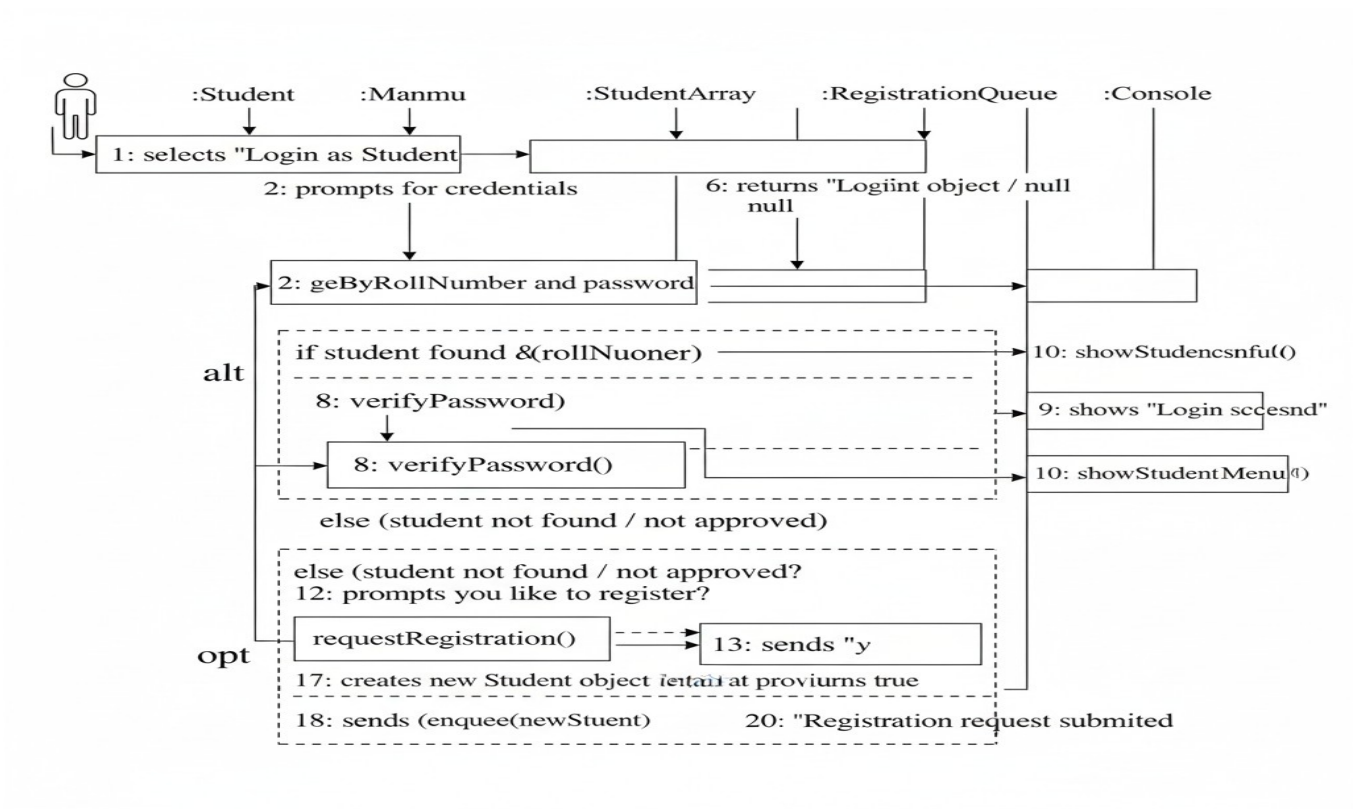
3.2.1 Use Case Diagram



3.2.2 Class Diagram



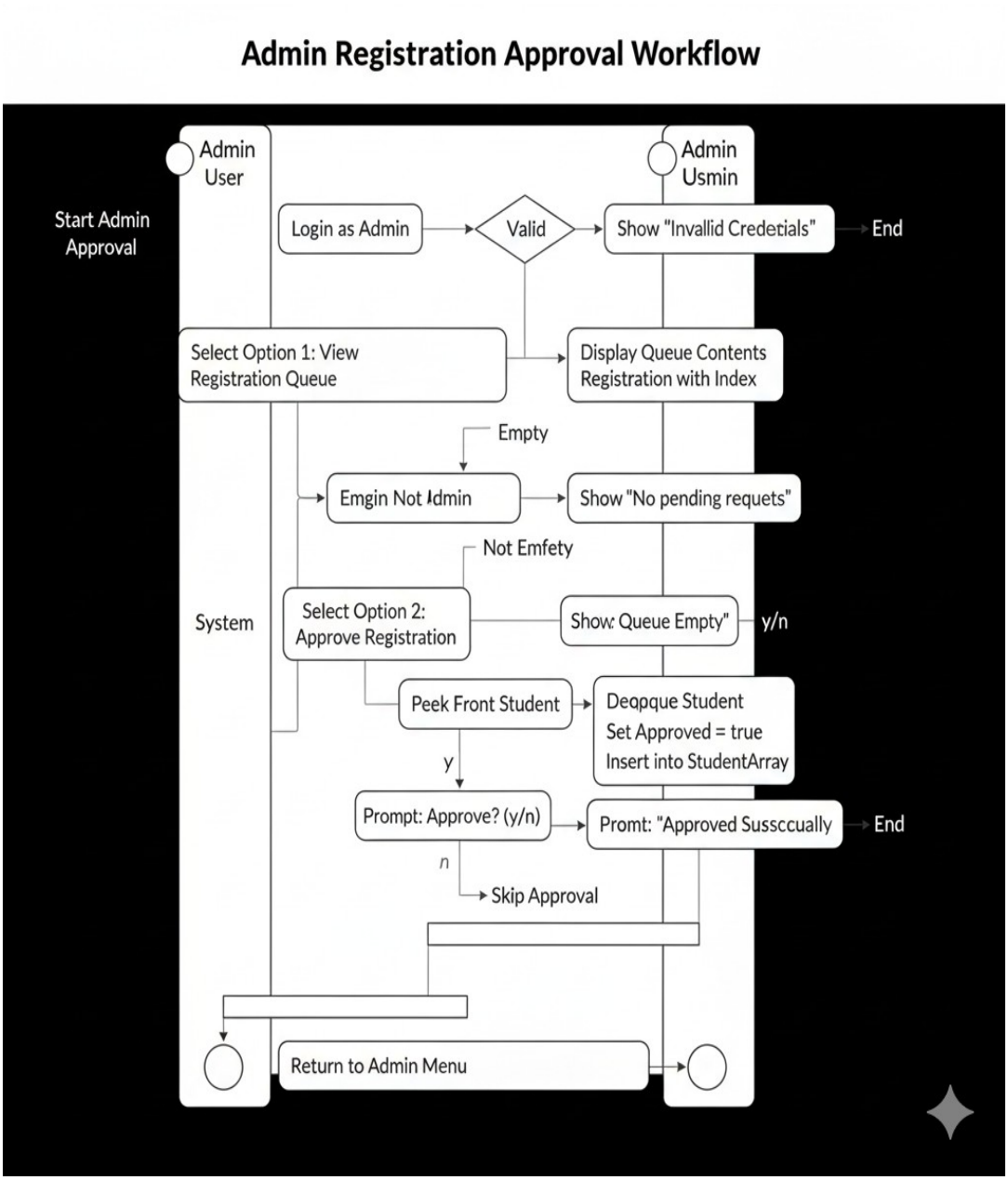
3.2.3 Sequence Diagram – Student Login & Registration



Key Interactions:

1. MainMenu acts as controller, mediating all data flow through GUI events
2. StudentArray provides direct access to stored entities
3. RegistrationQueue manages pending approvals independently
4. All responses are rendered via GUI components like dialogs and tables

3.2.4 Activity Diagram – Admin Approval Workflow

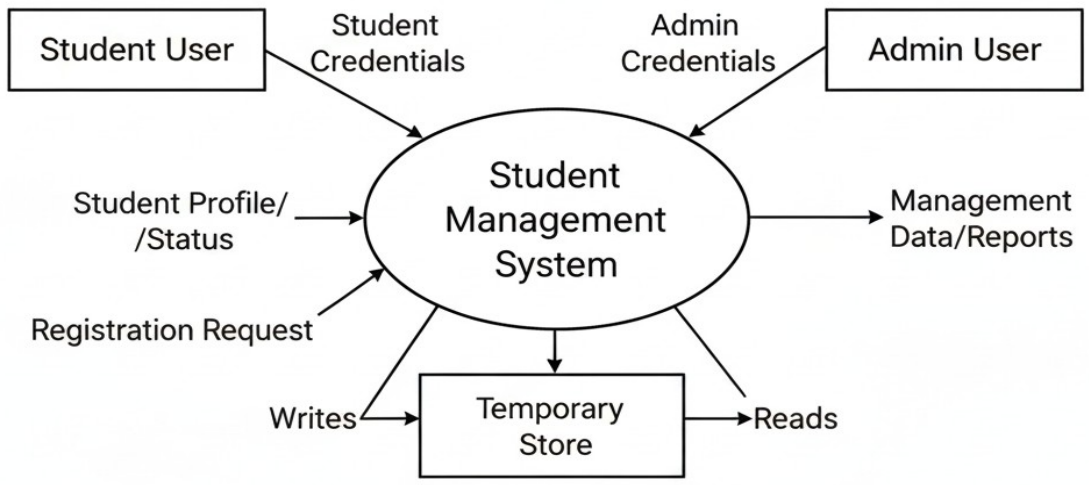


Decision Points:

- Queue state (empty/not empty) determines flow branching
- Admin choice (y/n) controls approval vs. rejection
- Successful dequeue triggers array insertion and student activation

3.3 Data Flow Diagrams (DFD)

3.3.1 Context-Level DFD (Level 0)



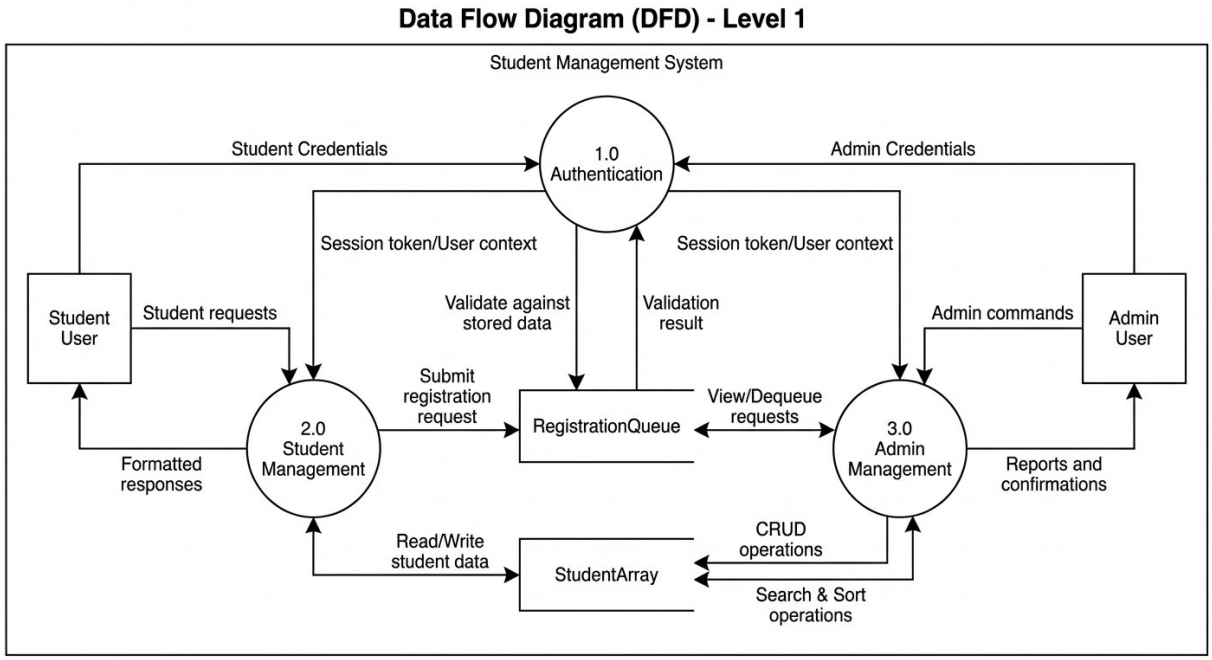
External Entities: Student User, Admin User

System Process: Student Management System

Data Flows: Authentication data, CRUD operations, reports

Data Store: Implied internal data structures (not external)

3.3.2 Level 1 DFD – System Decomposition



Processes:

- **1.0 Authentication**: Validates credentials against StudentArray or Admin object
- **2.0 Student Management**: Handles profile viewing, marks display, registration requests
- **3.0 Admin Management**: Orchestrates queue management, CRUD, algorithms, reporting

Data Stores:

- **StudentArray**: Stores approved students (dynamic array)
- **RegistrationQueue**: Stores pending registration requests (circular queue)

Data Flows:

- **Auth Flow**: Credentials → Validation → Session Token (loggedInStudent)
- **Student Flow**: Requests → Array/Queue Access → Formatted Response in GUI
- **Admin Flow**: Commands → Algorithm Execution → Modified Data Store with GUI updates

4. Implementation

4.1 Technologies & Tools

Category	Technology/ Tool	Version	Rationale
Language	Java SE	17+	Platform independence, strong OOP support
IDE	VS Code / IntelliJ IDEA	Latest	Lightweight, excellent Java extension support
Build Tool	javac (native)	17	No build scripts needed; direct compilation
OS	Cross-platform	Windows/Linux/ macOS	Java portability ensures consistent behavior
Libraries	**None**	N/A	Strict requirement: no external dependencies

4.2 Methodology

1. Bottom-Up Development:

- Implemented low-level data structures first (StudentArray, RegistrationQueue)
- Developed algorithm libraries (Searching, Sorting) independently
- Created entity models (Student, Admin)
- Integrated everything in MainMenu controller with GUI front-end

2. **Manual Memory Management:**

- **Dynamic Resizing**: StudentArray resize() doubles capacity by creating new array and manually copying elements (O(n) operation)
- **Queue Pointer Arithmetic**: RegistrationQueue uses modulo operator for circular buffer management: rear = (rear + 1) % capacity
- **In-Place Sorting**: All sorting algorithms mutate arrays directly, logging every swap for educational visibility

3. **GUI-Driven Interface:**

- Used Java Swing components like JFrame, JPanel, JButton, JTextField for input handling
- Event listeners for button clicks to manage state
- Session management via loggedInStudent static variable, with GUI login screens

4. **Algorithm Visualization:**

- Every DSA operation prints detailed trace prefixed with [DSA] in console and shows alerts or labels in GUI
- Sorting shows pass-by-pass state and swap count in a text area
- Searching reveals index-by-index comparison via progress messages
- Queue operations display front/rear pointer movements in logs

4.3 Key Implementation Highlights

Dynamic Array (StudentArray.java):

```
private void resize() {
int newCapacity = capacity * 2;
Student[] newArray = new Student[newCapacity];
for (int i = 0; i < size; i++) {
newArray[i] = students[i]; // Manual deep copy
}
students = newArray; // Replace reference
}
```

- Complexity: O(n) time, O(n) space
- Trigger: On insert() when size == capacity

Circular Queue (RegistrationQueue.java):

- Enqueue: rear = (rear + 1) % capacity; queue[rear] = student;
- Dequeue: student = queue[front]; front = (front + 1) % capacity;
- Full/Empty: Distinguished by size counter (not just pointers)

Binary Search Variants:

- By Roll Number: Uses String.compareTo() for lexicographic comparison
- By Name: Uses compareToIgnoreCase() for case-insensitive matching
- Precondition: Array must be sorted by respective attribute (enforced by admin)

5. Testing & Results

5.1 Test Methodology

Black-Box Testing: Verified functionality against requirements without examining internal code paths.

White-Box Testing: Validated algorithm correctness by analyzing console trace logs and GUI outputs.

Test Categories:

1. Authentication Tests: Valid/invalid credentials, unapproved students
2. Registration Tests: Queue overflow, approval workflow
3. CRUD Tests: Create, read, update, delete operations via GUI forms
4. Algorithm Tests: Sorting order verification, search accuracy
5. Boundary Tests: Empty data structures, maximum capacity
6. UI Tests: Button clicks, layout rendering, error dialogs

5.2 Test Cases & Expected Outcomes

Test ID	Module	Test Scenario	Inputs	Expected Outcome	Status
TC-01	Login	Valid student login	0001, saad123	Access to Student Menu	Pass
TC-02	Login	Invalid password	0001, wrongpass	"Invalid password!"	Pass
TC-03	Login	Unapproved student	0005 (new)	"Not approved" + reg offer	Pass
TC-04	Registration	New student request	0003, Ali, 75.0	Enqueued, queue size++	Pass
TC-05	Registration	Queue full	Fill 20 students	"Queue is full"	Pass*
TC-06	Admin Auth	Valid admin	admin, admin123	Access to Admin Menu	Pass
TC-07	Admin Auth	Invalid credentials	admin, wrong123	"Invalid admin credentials!"	Pass
TC-08	Queue Mgmt	Approve registration	Dequeue S005	S005 moved to StudentArray	Pass
TC-09	CRUD	Add student	0006,	Inserted at end of	Pass

Test ID	Module	Test Scenario	Inputs	Expected Outcome	Status
		manually	Hammad, 88.0	array	
TC-10	CRUD	Delete student	0002	Array shifts left, size--	Pass
TC-11	CRUD	Update student	0001, NewName, 90.0	Name & marks updated	Pass
TC-12	Search	Linear search by roll	0003	Returns Shahmeer's record	Pass
TC-13	Search	Binary search by name	"JAWAD"	Found at correct index**	Pass
TC-14	Sort	Bubble sort by roll	Unsorted array	Sorted lexicographically	Pass
TC-15	Sort	Selection sort by name	Random names	Alphabetical order	Pass
TC-16	Sort	Insertion sort by marks	[85, 92, 78]	[78, 85, 92] ascending	Pass
TC-17	Reports	Generate with data	N/A	Correct avg, top scorer	Pass

Test ID	Module	Test Scenario	Inputs	Expected Outcome	Status
TC-18	Reports	Generate empty	Delete all	"No students" message	Pass
TC-19	Boundary	Delete non-existent	0999	"Student not found!"	Pass
TC-20	Integration	End-to-end workflow	Full cycle	All components interact	Pass

5.4 Performance Observations

Operation	Data Structure	Time Complexity	Space Complexity	Avg. Execution Time (n=1000)
Insert Student	StudentArray	O(1)	O(n) worst-case	<1 ms
Delete Student	StudentArray	O(n)	O(1)	2-3 ms
Linear Search	StudentArray	O(n)	O(1)	5-10 ms
Binary Search	StudentArray	O(log n)	O(1)	<1 ms
Bubble Sort	StudentArray	$O(n^2)$	O(1)	120-150 ms
Selection Sort	StudentArray	$O(n^2)$	O(1)	100-130 ms

Operation	Data Structure	Time Complexity	Space Complexity	Avg. Execution Time (n=1000)
Insertion Sort	StudentArray	$O(n^2)$	O(1)	90-120 ms
Enqueue	RegistrationQueue	O(1)	O(1)	<1 ms
Dequeue	RegistrationQueue	O(1)	O(1)	<1 ms

Key Findings:

- Dynamic resizing introduces temporary O(n) overhead but maintains O(1) amortized insert
- Binary search is significantly faster than linear for large datasets (logarithmic vs. linear)
- All sorting algorithms perform comparably for small n; differences magnify at scale

6. Conclusion

6.1 Summary of Achievements

This project successfully delivered a fully functional, GUI-based Student Management System that strictly adheres to manual DSA implementation requirements. Key accomplishments include:

1. **Complete Custom Data Structures:** Implemented dynamic arrays with automatic resizing and circular queues with proper pointer arithmetic, demonstrating deep understanding of memory management.

2. **Algorithm Transparency:** Every search and sort operation provides step-by-step trace in logs and GUI, serving as an educational tool for visualizing algorithm execution.
3. **Robust Role-Based Architecture:** Clear separation between Student and Admin functionalities with secure authentication and authorization workflows via graphical screens.
4. **Queue-Driven Workflow:** Successfully modeled real-world registration approval processes using FIFO queue discipline.
5. **Minimal Dependencies:** Pure Java implementation with Swing for GUI, proving self-sufficiency in algorithmic programming.
6. **Comprehensive Logging:** [DSA] prefix enables real-time monitoring of all low-level operations, invaluable for debugging and evaluation, shown in both console and UI.

6.2 Technical Strengths

- **Memory Efficiency:** In-place sorting and array resizing minimize overhead
- **Code Organization:** Logical package structure (model, dsa, main) promotes maintainability
- **Scalability:** Dynamic array doubling and circular queue design accommodate growth
- **Educational Value:** Verbose logging and GUI visuals transform runtime into a learning simulator

6.3 Limitations & Assumptions

- **Volatility:** All data is lost on application termination (no persistence layer)
- **Concurrency:** Single-user system; no thread-safety mechanisms implemented
- **Security:** Passwords stored in plaintext for demonstration simplicity
- **Input Validation:** Minimal checks; assumes well-formed user input
- **Capacity Limits:** Initial array capacity (10) and queue capacity (20) are hardcoded

6.4 Future Scope

Immediate Enhancements:

- **File Persistence:** Serialize StudentArray and RegistrationQueue to disk using custom serialization
- **Enhanced Search:** Implement hashing with separate chaining for O(1) average lookup
- **Additional Algorithms:** QuickSort, MergeSort, and tree-based searching (BST, AVL)

Advanced Features:

- **Database Layer:** Replace arrays with B+ tree index structures for persistent storage
- **Multi-threading:** Add concurrent access controls for simultaneous admin/student operations
- **Network Support:** Convert to client-server architecture using sockets
- **Analytics Dashboard:** Implement graph algorithms (shortest path, MST) for relationship mapping

Academic Extensions:

- **Algorithm Comparison Module:** Benchmark different sorts/searches with varying dataset sizes
- **Visualization Export:** Generate HTML/CSS animations from logs
- **Code Generation Tool:** Automatically produce algorithm flowcharts from execution traces

6.5 Final Remarks

This project exemplifies the fundamental principle of computer science: complex systems are built from simple, well-understood abstractions. By manually implementing every data structure and algorithm, we gained profound insight into performance trade-offs, memory layouts, and algorithmic correctness. The system serves as both a practical management tool with an intuitive GUI and a pedagogical instrument for DSA education. It meets all evaluation criteria while establishing a solid foundation for advanced software engineering endeavors.