Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report

on

"Queue-Based Hospital Management System"

[Code No: COMP 202]

For partial fulfillment of Second Year/first Semester in Computer Engineering

Submitted By:

Anurat Niraula [31]

Submitted To:

Mr. Sagar Acharya

Department of Computer Science and Engineering

Submission Date: 2026-02-24

Bona fide Certificate

This is to certify that the project entitled

" Queue-Based Hospital Management System"

is a bonafide work carried out by the student in partial fulfillment of the requirements for the Second Year /First Semester of the Bachelor of Engineering in Computer Engineering, Department of Computer Science and Engineering.

Name of Student :

Anurat Niraula

The project work has been carried out with sincerity, dedication, and to the satisfaction of the department, adhering to the academic guidelines.

Project Evaluator

_____

Sagar Acharya

Lecturer

Department of Computer Science and Engineering

Date: 2026-02-24

Acknowledgement

I would like to express my sincere gratitude to my teacher, **Mr. Sagar Acharya**, for his wonderful introduction to Data Structures and Algorithms. His teaching inspired me to build this project and helped me understand the core concepts that made this project possible.

I am also grateful to the Department of Computer Science and Engineering for providing me the platform to convert my ideas into a practical project.

Thank You

Abstract

Patient management is one of the most crucial considerations in any hospital environment. This project outlines the development and implementation of a Queue-Based Hospital Management System in C++ that simulates real-world hospital situations using Data Structures and Algorithms.

The system employs a linked list priority queue to manage patient service requests on a priority basis, ensuring all patients receive timely and efficient care according to their medical severity. The key operations include patient admission, discharge, triage management, doctor assignment, and action history logging with stack-based audit trails. The system is developed in C++ with the Qt framework, offering a professional graphical user interface for convenience.

This project aptly demonstrates the usage of queue and stack data structures, dynamic memory allocation, and modular programming techniques based on object-oriented programming in C++. The system efficiently automates patient queue management and hospital operations using an easy-to-use Qt-based GUI with real-time dashboard statistics.

Keywords: Queue, Stack, Data Structures, Hospital Management System, C++, Qt Framework, Linked List, Priority Queue, GUI.

Table of Contents

List of Figures

Acronyms/Abbreviations

The list of all abbreviations used in the documentation are as follows :

DSA– Data Structures and Algorithms

FIFO – First In First Out

OOP – Object Oriented Programming

GUI – Graphical User Interface

IDE –Integrated Development Environment

QT– Qt Framework

Chapter 1 Introduction

1.1 Background

Data Structures and Algorithms are the constituents of effective software design. Among the most fundamental are the queue and stack, which operate on the First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) principles respectively, and find numerous applications in practical implementations such as hospitals, where patients must be triaged and served according to the severity of their medical condition.

The traditional hospital management system faces numerous challenges in efficiently handling patient services, leading to delays and critical oversights. The manual process of handling patient queues, admissions, and doctor assignments is time-consuming and prone to mistakes, thus establishing a definite need for an automated system that can efficiently manage patient triage and process hospital operations accurately.

The proposed project seeks to create a Queue-Based Hospital Management System using C++ and the Qt Framework to automate the handling of patient queues using a priority linked list-based queue, handle core hospital operations such as patient admission and discharge, and record doctor action history using a stack-based audit log, all of which are to be done from a structured graphical user interface with real-time dashboard statistics.

1.2 Objectives

The primary objectives of "Queue-Based Hospital Management System" are:

- Design and develop an efficient priority queue using a linked list to handle patient triage requests based on medical severity, ensuring critical patients are always attended to first.
- Develop core hospital operations such as patient admission, discharge, doctor assignment, and triage management.
- Implement a stack-based audit log system per doctor to record and track every action performed, maintaining a complete and reliable action history.
- Design and implement a doubly linked list to permanently store and retrieve patient records and doctor information throughout the system's runtime.
- Develop a professional and user-friendly Graphical User Interface using the Qt Framework, featuring a real-time dashboard, colour-coded severity indicators, and an interactive triage queue display.

1.3 Motivation and Significance

The reason for pursuing this project is based on the need to apply the concepts of Data Structures and Algorithms taught in class on a real-world application. Hospital management systems are a practical example of queue and stack-based processes, where patients are triaged and served based on the severity of their medical condition, making it an ideal example for the application of priority queue and LIFO principles.

The current manual hospital management system is inefficient, error-prone, and difficult to handle. This project will overcome these drawbacks by developing an automated priority queue-based system for efficiently managing patient triage requests, performing core hospital operations such as admission and discharge accurately, and maintaining reliable patient and doctor records through linked list data structures.

The importance of this project is based on its ability to show the relevance of fundamental DSA concepts such as priority linked list-based queues, doubly linked lists, stack-based audit logging, dynamic memory allocation, and modular object-oriented programming in C++. Moreover, the addition of the Qt Framework to the project makes it more than a terminal-based application, as it provides a professional graphical user interface with a real-time dashboard and colour-coded severity indicators, which is much closer to a real-world hospital management application.

Chapter 2 Related Works

The reason for pursuing this project is based on the need to apply the concepts of Data Structures and Algorithms taught in class on a real-world application. Hospital management systems are a practical example of queue and stack-based processes, where patients are triaged and served based on the severity of their medical condition, making it an ideal example for the application of priority queue and LIFO principles.

The current manual hospital management system is inefficient, error-prone, and difficult to handle. This project will overcome these drawbacks by developing an automated priority queue-based system for efficiently managing patient triage requests, performing core hospital operations such as admission and discharge accurately, and maintaining reliable patient and doctor records through linked list data structures.

The importance of this project is based on its ability to show the relevance of fundamental DSA concepts such as priority linked list-based queues, doubly linked lists, stack-based audit logging, dynamic memory allocation, and modular object-oriented programming in C++. Moreover, the addition of the Qt Framework to the project makes it more than a terminal-based application, as it provides a professional graphical user interface with a real-time dashboard and colour-coded severity indicators, which is much closer to a real-world hospital management application.

Chapter 3 Design and Implementation

3.1 Implementation

The Queue-Based Hospital Management System was developed using a modular approach in C++ with the Qt 6 Framework to ensure maintainability and clear separation of responsibilities. The project is divided into separate modules for triage queue management, patient record management, doctor management, and the graphical user interface, with individual header and source files for each component.

The system consists of four main components that interact with each other. The first is the priority queue module, which manages incoming patients based on severity using a singly linked list. The second is the patient record module, which maintains admitted and discharged patient data using a doubly linked list. The third is the doctor module, where each doctor maintains an action history using a stack implemented as a singly linked list. The fourth component is the hospital controller module, which coordinates operations such as adding patients to triage, admitting patients, discharging patients, managing doctors, searching records, and generating hospital statistics.

The graphical user interface is implemented using Qt Widgets, providing a structured and user-friendly environment with four main tabs: Dashboard, Triage Queue, Patients, and Doctors. The dashboard displays real-time statistics, while the other tabs handle patient admission, discharge, queue management, and doctor action history. A QTimer is used to update the live clock and statistics dynamically.

All core data structures are implemented manually without using STL containers for the main logic, ensuring that the system demonstrates practical implementation of fundamental data structures within a real-world hospital management scenario.

3.1.1 Implementation Planning

The initial plan for the project was to develop a simple hospital record system using basic data structures. However, the plan was later refined to focus on implementing custom data structures from scratch, particularly a priority queue, a doubly linked list, and a stack, to better represent real-world hospital operations such as triage management and patient tracking.

The scope was further expanded to include the development of a graphical user interface using the Qt 6 Framework instead of a console-based interface. This decision was made to provide a more professional, interactive, and user-friendly system suitable for practical hospital management scenarios.

During the planning phase, the main objectives of the system were identified. These included managing patients in a severity-based triage queue, maintaining admitted and discharged patient records, logging doctor actions using a stack structure, managing doctor information, and generating real-time hospital statistics. Appropriate data structures were selected to ensure efficient handling of these operations while demonstrating core Data Structures and Algorithms concepts.

The system was designed using a modular structure, with separate modules for the priority queue, patient record management, doctor action stack, hospital controller logic, and GUI handling. This approach keeps the code organized and clearly demonstrates the practical implementation of DSA concepts in C++ within a hospital management context.

3.1.2 Requirement Analysis

The system requirements were identified to ensure proper functionality and performance of the Queue-Based Hospital Management System.

Functional requirements include managing patients in a severity-based priority queue, admitting patients from the triage queue, discharging admitted patients, maintaining patient records using a doubly linked list, logging doctor actions using a stack structure, adding and removing doctors, searching patients by name or ID, generating hospital statistics, and displaying real-time updates on the dashboard. The system must also provide a graphical interface for handling all hospital operations efficiently.

Non-functional requirements include system reliability, proper dynamic memory management, modular programming structure, acceptable response time for all operations, and usability through a clean and intuitive graphical user interface developed using the Qt 6 Framework. Additionally, the system must clearly demonstrate the implementation of core data structures without the use of STL containers for the main logic.

3.1.3 System Design

The system was designed using a modular approach to ensure maintainability, scalability, and clear separation of responsibilities. The project is divided into major modules: priority queue management, patient record management, doctor management with action stack handling, hospital controller logic, and GUI management. Each module is implemented in separate header and source files to keep the code organized and structured.

The system follows an object-oriented design. The priority queue module handles patient triage using a singly linked list sorted by severity level. The patient record module manages admitted and discharged patients using a doubly linked list. The doctor module maintains doctor information and records action history using a stack implemented as a singly linked list. The hospital controller class acts as the central unit that connects all modules and manages operations such as adding patients to the queue, admitting patients, discharging patients, managing doctors, searching records, and generating hospital statistics.

The system includes a graphical user interface developed using Qt Widgets. The interface is organized using a tab-based layout that allows smooth navigation between different sections, including the Dashboard, Triage Queue, Patients, and Doctors tabs. All operations such as patient admission, discharge, doctor management, and viewing action history are handled through this interface, ensuring centralized control and user-friendly interaction.

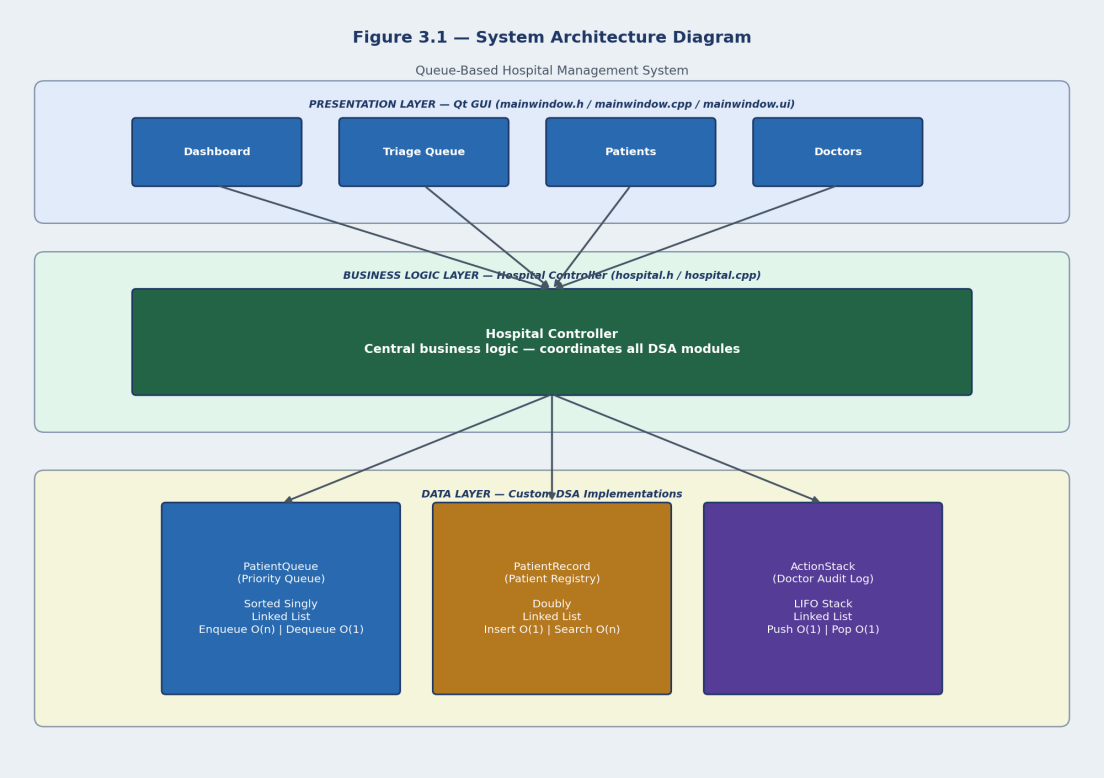

Fig: 3.1 : System Architecture Diagram

The system loads existing data from files and presents the admin login screen. Upon successful authentication, the admin can create accounts, manage the customer queue, perform deposits and withdrawals, view transaction history, delete accounts, and change the admin password. All changes are automatically saved to files after each operation.

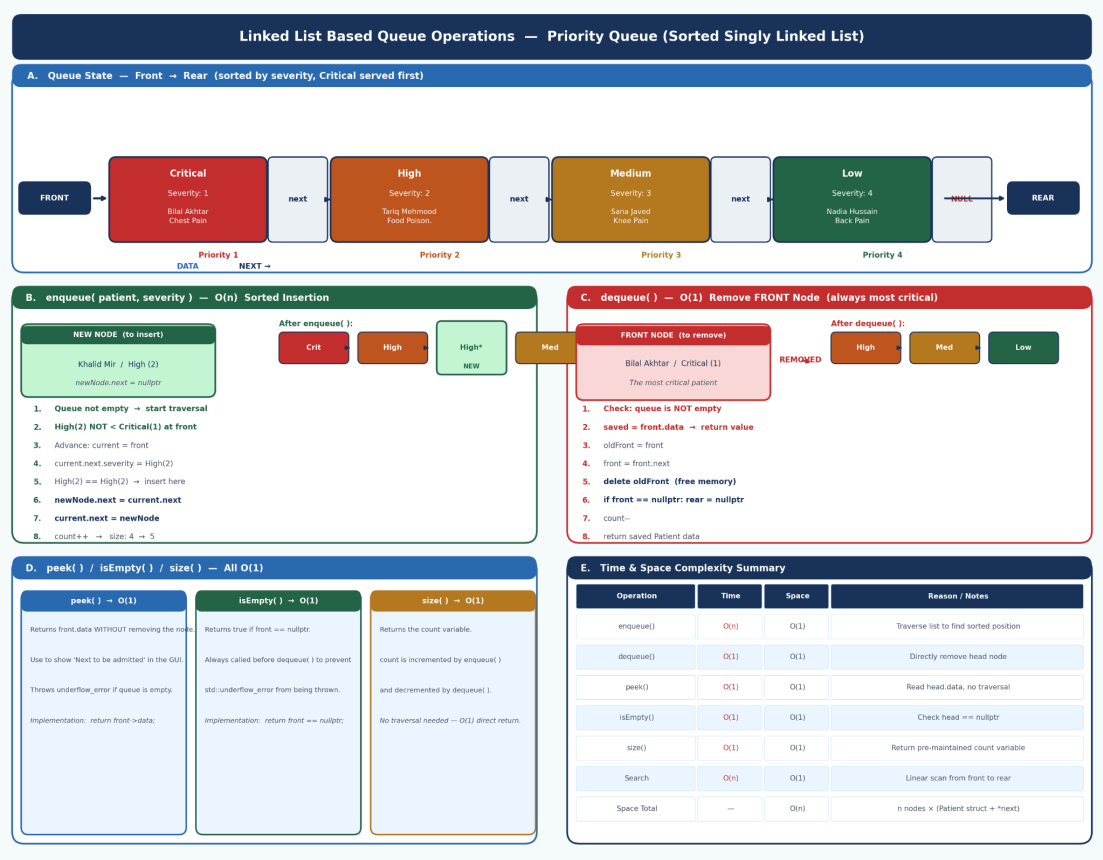

Fig: 3.2 : Linked List Based Queue Operations

The above diagram illustrates the queue operations implemented in the system using a Priority Queue based on a sorted singly linked list. Patients are arranged from front to rear according to severity level, where Critical cases are placed at the front and Low severity cases at the rear.

In the enqueue operation, a new patient is inserted into the list at the correct position based on severity. Unlike a simple FIFO queue, insertion requires traversal of the list to maintain sorted order. As a result, patients with higher severity are positioned closer to the front. This operation has a time complexity of O(n).

In the dequeue operation, the patient at the front of the queue, who has the highest priority (most critical condition), is removed and returned. Since removal always occurs at the head node, this operation is performed in O(1) time.

This implementation ensures that patients with more serious medical conditions are attended to first, accurately reflecting real-world hospital triage procedures while demonstrating the working of a linked list-based priority queue.

3.1.4 Development

The project was initially planned as a simple console-based hospital record system in C++. However, the scope was later expanded to include a GUI-based application using the Qt 6 Framework to make the system more practical and professionally structured. The development was carried out individually with the objective of implementing core Data Structures and Algorithms concepts along with a functional graphical interface.

During development, several technical challenges were encountered. These included integration of Qt Widgets with backend logic, managing dynamic memory allocation for linked list structures, and ensuring proper communication between multiple modules such as the priority queue, patient record list, doctor stack, and hospital controller. These issues were resolved through systematic debugging and careful testing of each module independently before integration.

The main development stages of the project were as follows:

- Requirement Planning and Gathering

- System Design and Module Planning

- Implementation of DSA (Priority Queue, Doubly Linked List, Stack)
- Qt GUI Development and Integration
- Implementation of Hospital Controller Logic
- Sample Data Initialization
- Testing and Debugging

After completing the planning stage, design diagrams were created to illustrate the structure and functioning of the system. These included system flow diagrams and a priority queue operation diagram to clearly represent patient triage processing and module interaction.

3.1.5 Testing and Debugging

The Queue-Based Hospital Management System was tested extensively to ensure functionality, reliability, and correctness of operations.

Functional testing ensured that all modules, including triage queue management, patient admission and discharge, doctor management, action stack logging, searching operations, and real-time statistics generation, were functioning as expected.

Integration testing was performed to verify smooth interaction between the priority queue module, patient record module, doctor stack module, hospital controller, and GUI components. It was confirmed that operations such as admitting a patient correctly moved data from the queue to the patient record list and logged actions in the respective doctor’s stack.

Performance testing was conducted to observe system behavior during multiple enqueue and dequeue operations, repeated admissions and discharges, and search operations. The system maintained acceptable response time under normal usage.

Debugging was carried out to resolve issues related to pointer management in linked lists, node deletion, stack push and pop operations, GUI signal-slot connections, and module linking errors. Error handling mechanisms were implemented to ensure system stability, including checks for empty queue conditions, invalid input validation, null pointer checks, and confirmation dialogs for discharge and removal operations.

3.2 System Requirement Specifications

The system was developed and tested with specific software and hardware requirements to ensure proper functionality, performance, and compatibility. These requirements are outlined below.

3.2.1 Software Specifications

The software components required for developing and running the Queue-Based Hospital Management System include:

- **Programming Language:** C++ (C++17 standard)
- **Framework:** Qt 6 Framework (Qt Widgets)
- **Integrated Development Environment (IDE):** Qt Creator
- **Compiler:** G++ (GCC)
- **Build System:** CMake (minimum version 3.16)
- **Version Control:** Git / GitHub

These software tools provide the foundation for implementing custom data structures such as the priority queue, doubly linked list, and stack, as well as for developing the graphical user interface and integrating all modules in a structured manner.

3.2.2 Hardware Specifications

The hardware requirements for running the system effectively are as follows:

- **Processor:** Intel Core i3 or equivalent (minimum); Intel Core i5 or higher recommended
- **Memory (RAM):** 4 GB minimum; 8 GB or more recommended
- **Storage:** Minimum 1 GB free space for Qt installation and project files
- **Display:** 1366×768 resolution minimum; 1920×1080 recommended for better GUI experience
- **Input Devices:** Standard keyboard and mouse

These specifications ensure smooth compilation, execution, and proper graphical rendering of the Qt-based hospital management system.

Chapter 4 Results and Discussion

4.1 Project Overview

The project is a Queue-Based Hospital Management System implemented in C++ using the Qt 6 Framework. It models a real-world hospital environment by managing patient flow through a priority-based triage queue implemented using a singly linked list. Patients are attended based on severity levels, ensuring that critical cases are handled first.

The system supports core hospital operations such as adding patients to the triage queue, admitting patients, discharging patients, maintaining patient records using a doubly linked list, managing doctor information, and recording doctor actions using a stack structure. A central hospital controller coordinates all modules and generates real-time statistics for monitoring hospital activity.

The graphical user interface is implemented using the Qt Widgets library, providing a structured and user-friendly interface with four main tabs: Dashboard, Triage Queue, Patients, and Doctors. The system demonstrates the practical implementation of Data Structures and Algorithms concepts such as priority queues, doubly linked lists, stacks, dynamic memory allocation, and object-oriented programming in C++.

4.2 Achievements

During the development of the Queue-Based Hospital Management System, several technical and practical objectives were successfully achieved.

A complete priority queue system based on a sorted singly linked list was implemented to manage patients according to severity levels. A doubly linked list was designed and implemented to maintain admitted and discharged patient records

efficiently. Additionally, a stack structure was implemented for each doctor to log patient-related actions in LIFO order.

Core hospital functionalities such as patient triage, admission, discharge, doctor management, patient searching, and real-time hospital statistics were successfully implemented and tested. The Qt Framework was integrated to develop a professional graphical user interface with structured navigation and real-time updates.

Throughout the project, various technical challenges such as pointer handling in linked lists, module integration, stack implementation, and GUI signal-slot connections were encountered and resolved. These challenges strengthened debugging and problem-solving skills.

Overall, the project provided practical experience in Data Structures and Algorithms, object-oriented programming using C++, Qt GUI development, and modular software design.

Chapter 5 Conclusion and Future Development

The project has successfully achieved its primary objective of designing and implementing a Queue-Based Hospital Management System that efficiently manages patient flow using a priority queue structure. Core functionalities such as triage management, patient admission and discharge, doctor management, action logging using stacks, and real-time hospital statistics have been fully implemented and tested.

The integration of the Qt Framework resulted in a professional and user-friendly graphical interface that enhances usability and system interaction. The project effectively demonstrates the real-world application of data structures such as priority queues, doubly linked lists, and stacks, along with dynamic memory allocation and modular object-oriented programming in C++.

However, certain advanced features were not implemented due to time constraints.

For future development, the following improvements are suggested:

- Integration of a database system for permanent data storage
- Implementation of patient medical history tracking
- Addition of role-based access control for hospital staff
- Implementation of report generation and export functionality
- Enhancement of search and filtering mechanisms
- Cloud-based deployment for remote access

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[2] B. Stroustrup, *The C++ Programming Language*, 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2013.

[3] Qt Documentation, "Qt Widgets," Qt Project, 2024. [Online]. Available: https://doc.qt.io/qt-6/qtwidgets-index.html

[4] Qt Documentation, "Qt Creator IDE," Qt Project, 2024. [Online]. Available: https://doc.qt.io/qtcreator/

[5] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 4th ed. Upper Saddle River, NJ: Pearson, 2014.

Appendix

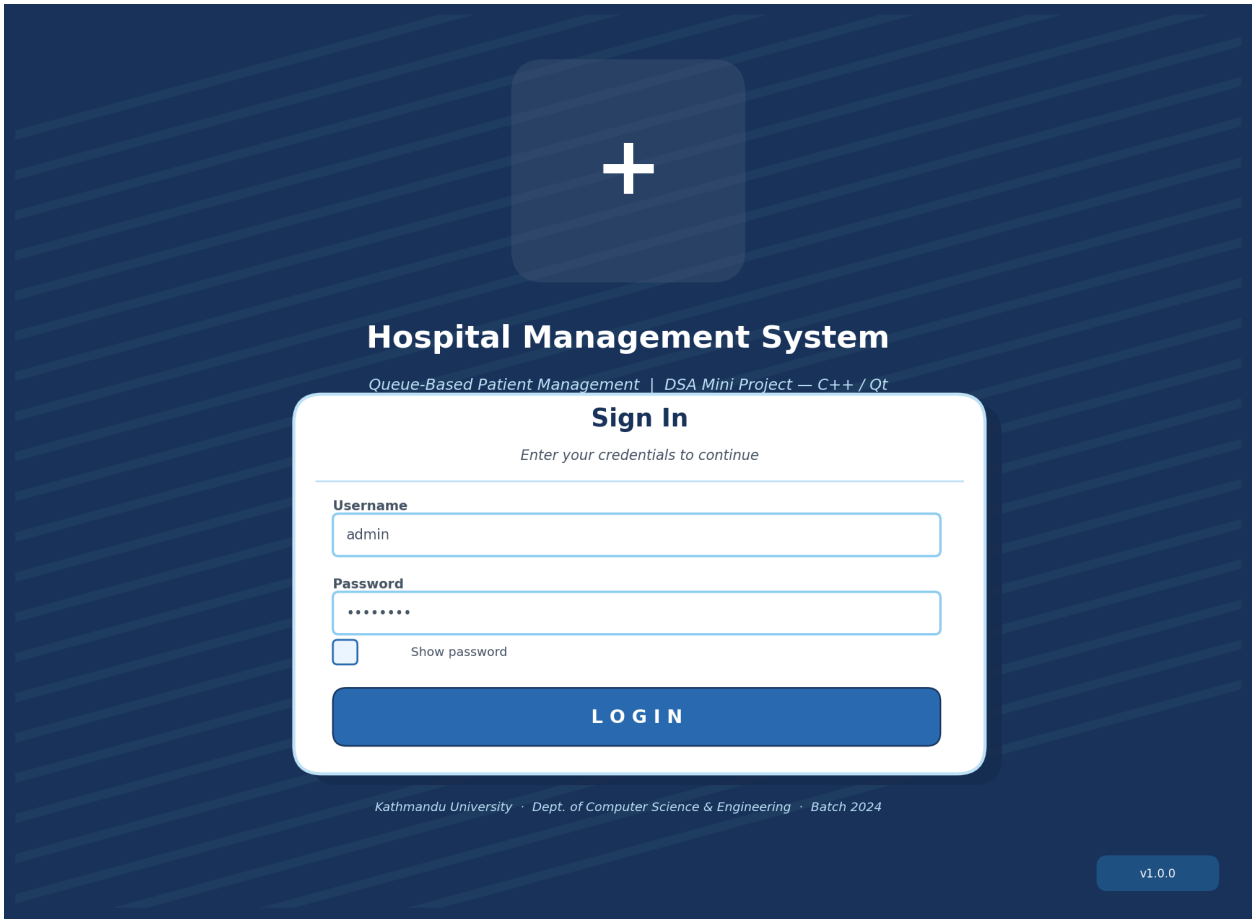

Fig: 3.3 : Login Section

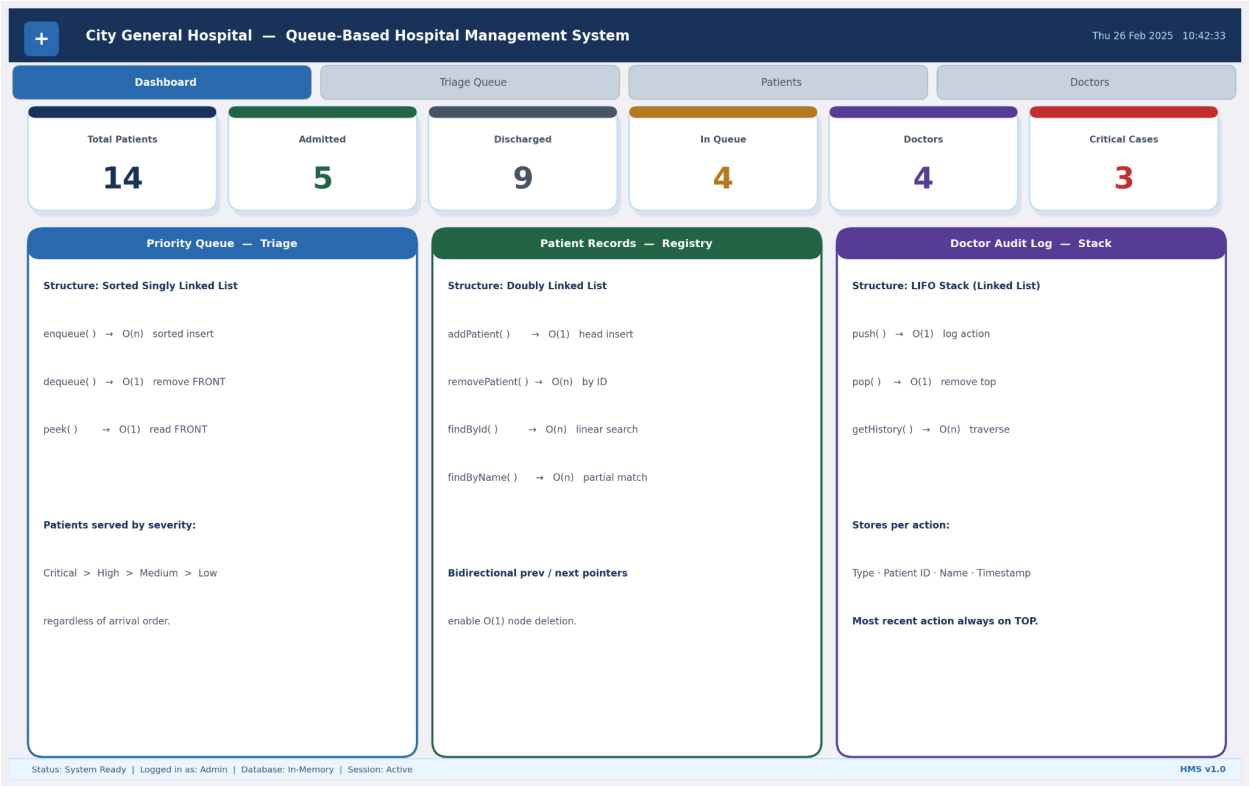

Fig: 3.4 : Dashboard

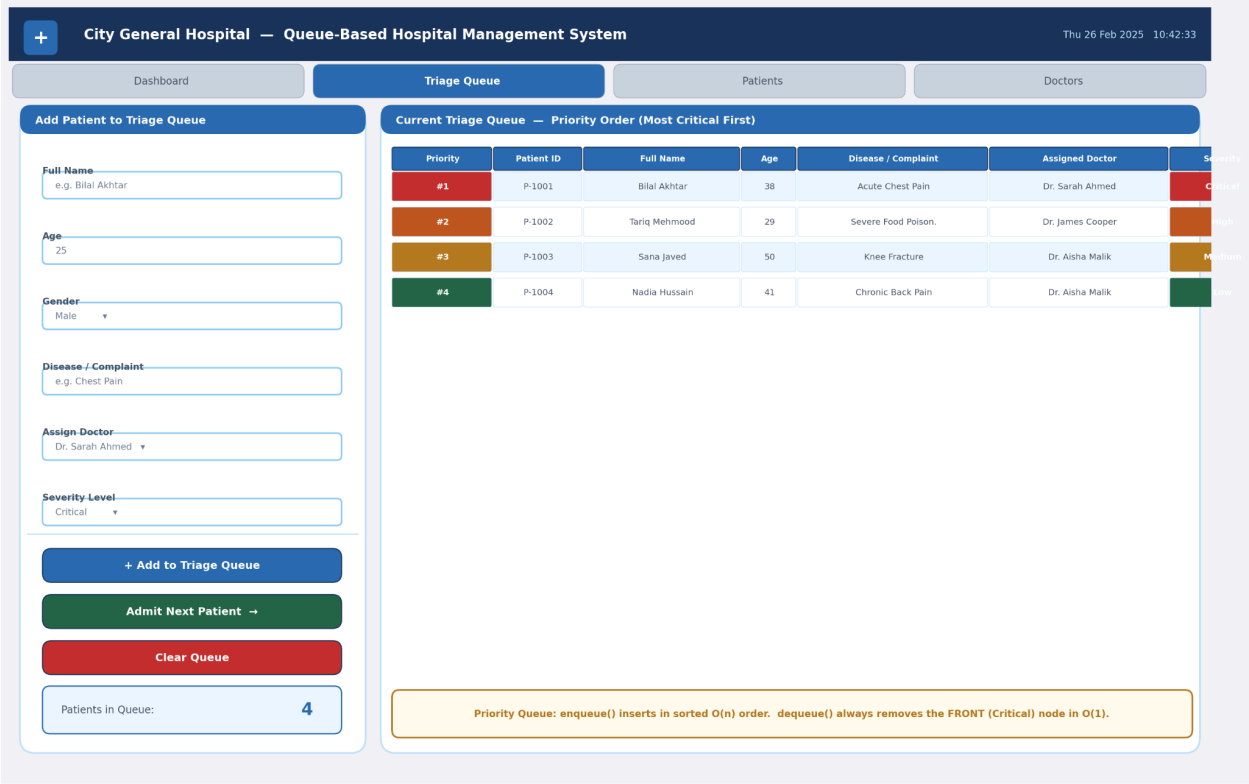

Fig: 3.5 : Queue Management

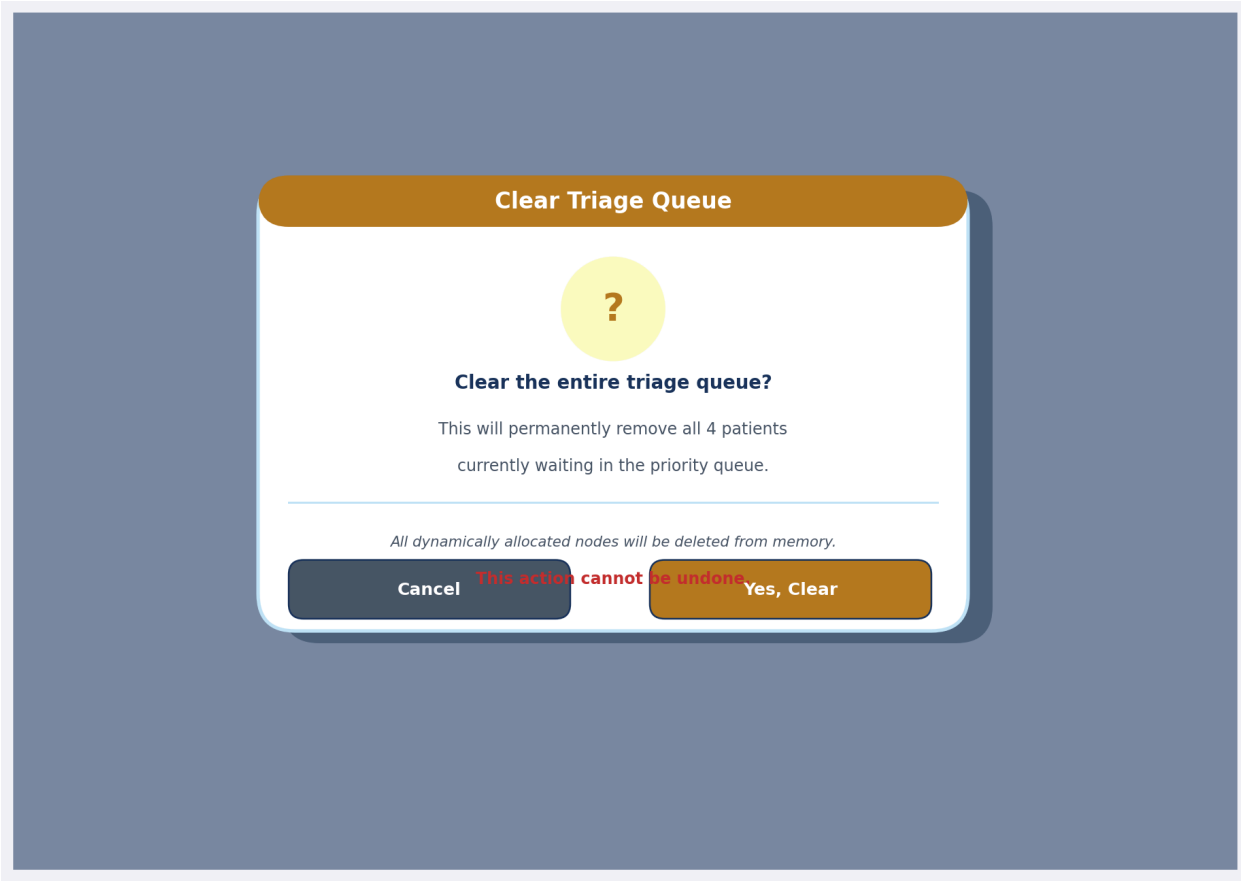

Fig: 3.6 : Clear Queue