



**Module Code & Module Title**

**CS5003NI Data Structures and Specialist Programming**

**30% Individual Coursework 1**

**Submission: Milestone 1**.

**Academic Semester: AY 2025/2026**

**Credit: 30 credit year long module**

**Student Name: Anjal Bhattarai**

**Project Title: Prison** **Management System**

**London Met ID: 24046565**

**College ID: NP01AI4A240091**

**Assignment Due Date: 26/12/2025**

**Assignment Submission Date: 26/12/2025**

**Submitted To: Subarna Sapkota**

|  |  |
| --- | --- |
| **GitHub Link** | **https://github.com/anjalbhattarai79/Prison-management-system.git** |

*I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

**TURNITIN REPORT**

**Table of Contents**

[1. Introduction 1](#_Toc217664239)

[1.2 Aim and Objectives 3](#_Toc217664240)

[1.3 Problem Statement 4](#_Toc217664241)

[2. MVC Architecture & ANT Setup 6](#_Toc217664242)

[3. Class Diagrams 9](#_Toc217664243)

[4. Wireframe Design and User Interface Screens 11](#_Toc217664244)

[5. CRUD Functionality 16](#_Toc217664245)

[5.1 Create Functionality 16](#_Toc217664246)

[5.2 Read Functionality 17](#_Toc217664247)

[5.3 Update Functionality 18](#_Toc217664248)

[5.4 Delete Functionality 20](#_Toc217664249)

[6. Test Cases 22](#_Toc217664250)

[Test Case 01 : TC\_CREATE\_PRISONER\_EMPTY\_NAME\_VALIDATION 22](#_Toc217664251)

[Test Case 02: TC\_CREATE\_PRISONER\_NEGATIVE\_AGE\_VALIDATION 24](#_Toc217664252)

[Test Case 03: CREATE\_PRISONER\_NEGATIVE\_SENTENCE\_VALIDATION 25](#_Toc217664253)

[Test Case 04: CREATE\_PRISONER\_INVALID\_DATE\_FORMAT\_EXCEPTION 26](#_Toc217664254)

[7. Conclusion 32](#_Toc217664255)

[8. Future Works 33](#_Toc217664256)

[9. Bibliography 35](#_Toc217664257)

[10. Appendix 36](#_Toc217664258)

**Table of Figures**

[Figure 1: MVC Architechture 7](#_Toc217664207)

[Figure 2: MVC Folder structure for Prison Management System 7](#_Toc217664208)

[Figure 3: Ant Setup (1) 8](#_Toc217664209)

[Figure 4: Ant Setup (2) 8](#_Toc217664210)

[Figure 5: Class diagram 9](#_Toc217664211)

[Figure 6: Wireframe of Home page 11](#_Toc217664212)

[Figure 7: Actual Home page 11](#_Toc217664213)

[Figure 8: Wireframe of Admin Login page 12](#_Toc217664214)

[Figure 9: Actual Admin login page 12](#_Toc217664215)

[Figure 10: Wireframe of Admin Dashboard 13](#_Toc217664216)

[Figure 11: Actual Admin Login Dashboard 13](#_Toc217664217)

[Figure 12: Wireframe of Family login portal 14](#_Toc217664218)

[Figure 13: Wireframe of Family Dashboardp 15](#_Toc217664219)

[Figure 14: Dialog Box to provide new prisoner detials 16](#_Toc217664220)

[Figure 15: New Prisoner being added 16](#_Toc217664221)

[Figure 16: Create new Prisoner Successfully 17](#_Toc217664222)

[Figure 17: Detail over view of all Prisoner 17](#_Toc217664223)

[Figure 18: Editable current details of prisoner when clicked edit button 18](#_Toc217664224)

[Figure 19: Status changed from active to released 18](#_Toc217664225)

[Figure 20: Updated details of prisoners 19](#_Toc217664226)

[Figure 21: DIalog Box to confirm deletion of prisoner details 20](#_Toc217664227)

[Figure 22: Deletion Confirmation Pop-up 20](#_Toc217664228)

[Figure 23: Prisoner with id 106 deleted 20](#_Toc217664229)

[Figure 24: Recently deleted prisoner stored in STACK 21](#_Toc217664230)

[Figure 25: Restore Confirmation 21](#_Toc217664231)

[Figure 26: Restored Deleted Prisoner and Stored back in QUEUE 21](#_Toc217664232)

[Figure 27: Add new prisoner with empty name 22](#_Toc217664233)

[Figure 28: Empty name error message displayed 23](#_Toc217664234)

[Figure 29: Error message for negative age 24](#_Toc217664235)

[Figure 30: Error message for negative sentence period 25](#_Toc217664236)

[Figure 31: Add Prisoner with invalide Date format 26](#_Toc217664237)

[Figure 32: Invalid Date Format Error message 27](#_Toc217664238)

**Table of Tables**

No table of figures entries found.

# Introduction

Prison administration is an inherently complex task that requires careful management of dynamic records, fairness in judicial processing, and adaptability to fluctuating inmate populations. In many countries, including Nepal, the scale of these challenges is compounded by systemic issues such as overcrowding, inadequate facilities, and inefficient case management. For example, the combined capacity of Nepal’s prison system is approximately 16,556 inmates, yet official data shows more than 24,000 prisoners currently housed nationwide, reflecting overcrowding of roughly 46.56% above capacity (Kathmandu School of Law Review, 2022). The failure to balance prison capacity with population has profound consequences. Overcrowded conditions are linked with heightened interpersonal conflict, degraded living environments, limited access to health services, and a chronic strain on administrative resources. Recent reports identify ongoing tensions and violent clashes in Nepal’s prisons as direct outcomes of severe inmate congestion and inadequate supervision (The Kathmandu Post, 2023). These realities illustrate the need for more effective information systems that assist administrators in recordkeeping, monitoring trends, and enabling timely operational decisions.

Within this real-world context, the Prison Management System project has been developed as a comprehensive educational software application for the CS5005 Data Structures and Specialist Programming module. The principal aim of this project is to bridge the gap between abstract theoretical learning and its application in practical, complex scenarios. By situating data structure and algorithm (DSA) principles within the domain of prison information management, the project demonstrates how foundational computer science constructs can be utilized to solve dynamic data challenges effectively.

Prison environments inherently involve data that changes frequently, such as admissions, releases, transfers, and legal status updates. To support this dynamism, the system uses Linked Lists to store prisoner records, allowing efficient insertion and deletion operations without fixed size constraints. Queues implement tracking of recent administrative activities under the First-In-First-Out (FIFO) paradigm, reflecting realistic event ordering. Stacks are incorporated to facilitate undo operations according to Last-In-First-Out (LIFO) behavior, demonstrating practical recovery mechanisms. Furthermore, the project employs both Linear Search (O(n)) and Binary Search (O(log n)) algorithms, enabling learners to contrast sequential and divide-and-conquer strategies within a real dataset context. Sorting operations demonstrate algorithmic complexity through organized arrangements based on multiple attributes such as ID, age, and admission date.

From a software engineering standpoint, the system embraces the Model-View-Controller (MVC) architectural pattern to ensure modularity, maintainability, and clarity. Data models encapsulate prisoner information, Java Swing interfaces present user interactions, and controller components manage validation and logic operations. This design adheres to single responsibility principles and reflects industry’s best practices for scalable software systems.

Although developed for academic purposes, the project also holds broader relevance to evolving prison management needs. Nepal’s prison system continues to grapple with structural challenges ranging from poorly maintained infrastructure to insufficient health services and case tracking mechanisms. A robust information system, even in prototype form, offers insights into how dynamic data handling, structured records, and algorithmic operations can contribute to more transparent and responsive administrative practices.

In summary, the Prison Management System serves a dual purpose; it is both a pedagogical tool that concretizes DSA concepts through realistic applications and a conceptual example of how software solutions can support complex institutional challenges. By aligning theoretical knowledge with disciplined software design, the project reinforces essential competencies in algorithmic thinking, architectural planning, and data-driven decision support bringing competencies critical for both academic assessment and future professional development.

## Aim and Objectives

**AIM**

The aim of this project is to design and implement an educational Prison Management System that demonstrates the practical application of core Data Structures and Algorithms concepts in solving real-world data management problems within a realistic administrative context .

**Objectives**

* Apply fundamental data structures such as Linked Lists, Queues, and Stacks to model and manage dynamic prisoner records in a realistic administrative system.
* Design and implement a structured CRUD-based prisoner management system with comprehensive input validation to ensure accuracy, consistency, and data integrity.
* Analyze and Compare Linear Search and Binary Search algorithms by integrating them into the system to enable efficient retrieval of prisoner information.
* Organize and Optimize prisoner records using appropriate sorting techniques like Selection Sort, Insertion Sort and Merge Sortbased on multiple attributes to enhance data accessibility and operational efficiency.
* Develop and evaluate a user-friendly graphical interface with recovery mechanisms that minimize user error and support safe restoration of deleted records.

## Problem Statement

Prison administration in developing countries such as Nepal faces persistent challenges due to outdated and inefficient record-management practices. Many prison facilities continue to rely on manual or semi-digital systems that are prone to human error, data inconsistency, and delayed information retrieval. These limitations are intensified by severe overcrowding, with Nepal’s prisons operating significantly beyond their intended capacity, placing excessive strain on administrative processes (Kathmandu School of Law Review, 2022). Under such conditions, maintaining accurate prison records and accessing information promptly during legal proceedings, medical emergencies, or security incidents becomes increasingly difficult.

Existing systems also lack the flexibility required to manage highly dynamic prison data. Frequent admissions, releases, transfers, and status updates demand efficient data handling mechanisms; however, traditional systems often employ rigid data structures and inefficient search techniques. As a result, routine operations such as locating prison records or generating administrative summaries rely on sequential scanning, leading to performance degradation as data volume increases. Furthermore, many legacy systems are architecturally monolithic, combining data handling, logic, and presentation layers in tightly coupled designs. This lack of modularity limits maintainability, hinders scalability, and increases the likelihood of errors during system updates. Inadequate input validation further compromises data integrity, allowing inaccurate or incomplete information to propagate through administrative workflows.

**Proposed System and Addressed Limitations**

The proposed Prison Management System addresses these limitations through structured application of data structures, algorithms, and modular software architecture. Dynamic prisoner records are managed using Linked Lists, enabling efficient insertion and deletion operations that align with the continuously changing nature of prison populations. To improve data retrieval efficiency, the system incorporates both Linear Search and Binary Search algorithms, reducing search time for sorted datasets and improving operational responsiveness.

The adoption of the Model-View-Controller (MVC) architectural pattern ensures clear separation of concerns between data models, business logic, and user interface components. This modular design enhances maintainability, simplifies future enhancements, and supports independent testing. Additionally, comprehensive input validation mechanisms ensure data accuracy and consistency, while Queue and Stack implementations support activity tracking and undo functionality, improving usability and administrative transparency.

Overall, the proposed system demonstrates how disciplined software design and algorithmic efficiency can overcome the operational limitations of traditional prison management systems. It serves both as an effective educational implementation of data structure concepts and as a conceptually viable solution for improving administrative reliability in resource-constrained institutional environments.

# MVC Architecture & ANT Setup

The Model-View-Controller (MVC) architecture is a software design pattern that separates an application into three interconnected components, promoting organized code structure, maintainable development, and scalability (Gamma, et al., 1994 ). In our Prison Management System, we have implemented this architecture to ensure clear separation of concerns and to support efficient data handling.

**Architecture Overview**

**Model**

The Model component represents the data and business logic layer. In our system, the PrisonerModel class encapsulates all prisoner-related data, including personal information (name, age, gender, address), crime details (type, description), sentence information (admission date, duration, release date), and status indicators (health status, incarceration status). This layer maintains data integrity and defines the structure of prisoner records, operating independently of the user interface to ensure consistent data logic regardless of presentation (Anon., n.d.).

**View**

The View component handles the presentation layer and user interface. Our system uses Java Swing components such as JTable, JOption (Anon., n.d.)Pane, and UI panels to display prisoner information, search results, and statistical data. The View renders data for users in a readable format and captures user inputs through forms and interactive elements. It communicates with the Controller to retrieve data for display but contains no business logic itself (GeeksforGeeks, 2023).

**Controller**

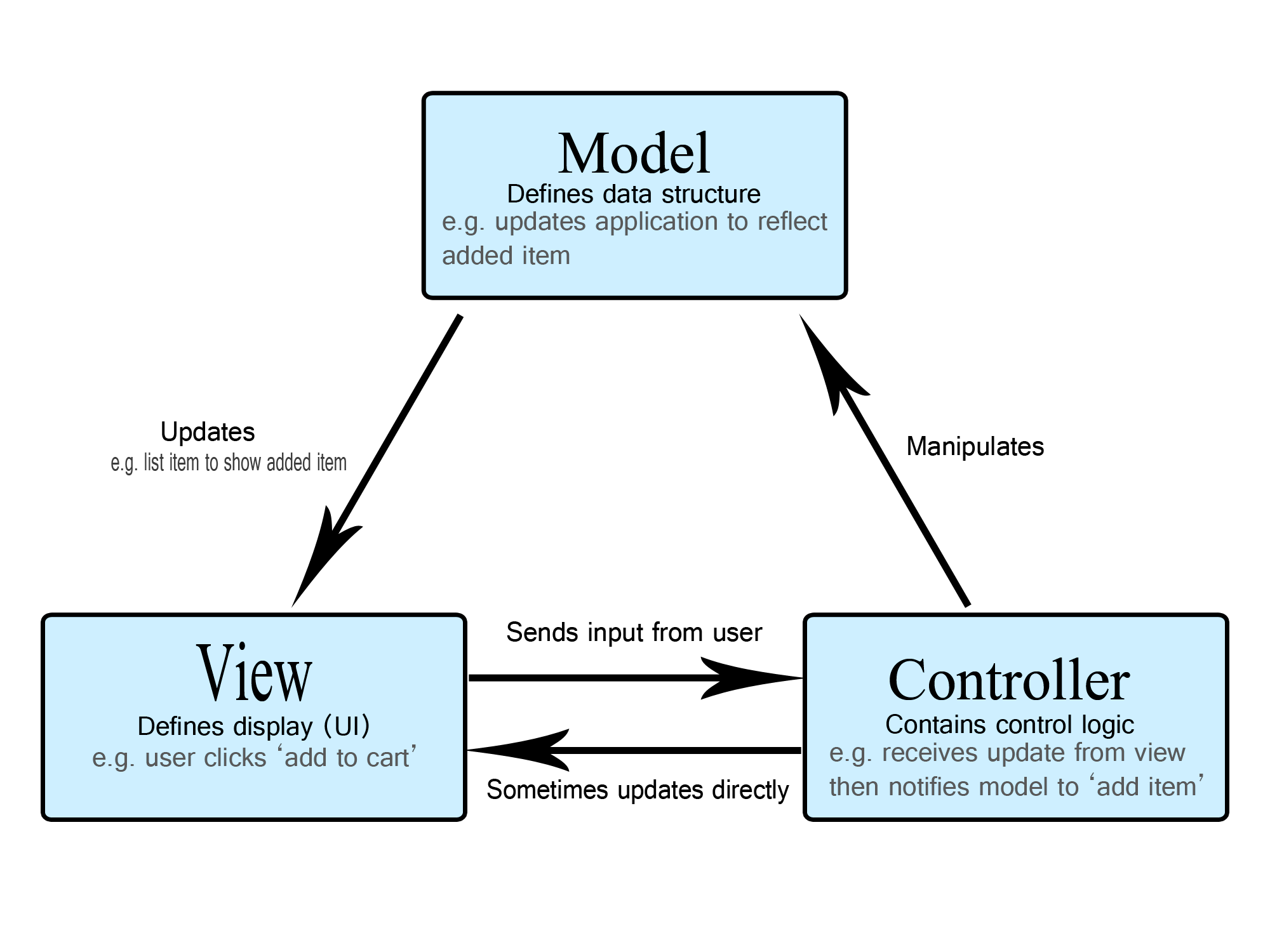
The Controller, exemplified by our PrisonController class, acts as an intermediary between the Model and View. It processes user requests, manipulates data through business logic operations (delegated to specialized classes like CRUD, SearchOperation, SortOperation, and TrashBinOperation), and updates the View accordingly. The controller manages data structures including a LinkedList for active prisoners, a Queue for recently added prisoners (demonstrating FIFO operations), and a Stack for deleted prisoners in the trash bin (demonstrating LIFO operations). It orchestrates complex operations like adding, updating, deleting, searching, and sorting prisoners while maintaining data consistency across collections (Gamma, et al., 1994 ).

Figure 1: MVC Architechture

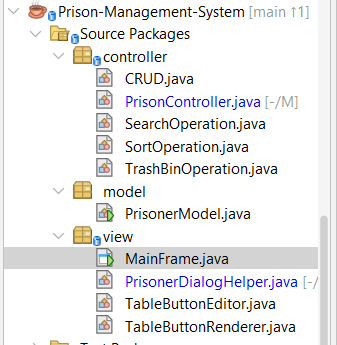


Figure 2: MVC Folder structure for Prison Management System

**ANT SETUP**

To compile and run the Prison Management System efficiently, the project uses Apache Ant, a Java-based build tool that automates compilation, packaging, and execution tasks. Ant allows developers to:

1. Automate Compilation: Compile all Java classes in the correct order, handling dependencies automatically.
2. Package Applications: Bundle compiled classes and resources into JAR files for easier distribution.
3. Manage Tasks: Clean previous builds, run tests, and execute the main program with predefined targets.

Using Ant streamlines project setup, reduces human errors during manual compilation, and ensures consistent builds across development environments. For academic submissions, it also demonstrates good software engineering practices by separating code, resources, and build logic (Apache Ant Project, n.d.).

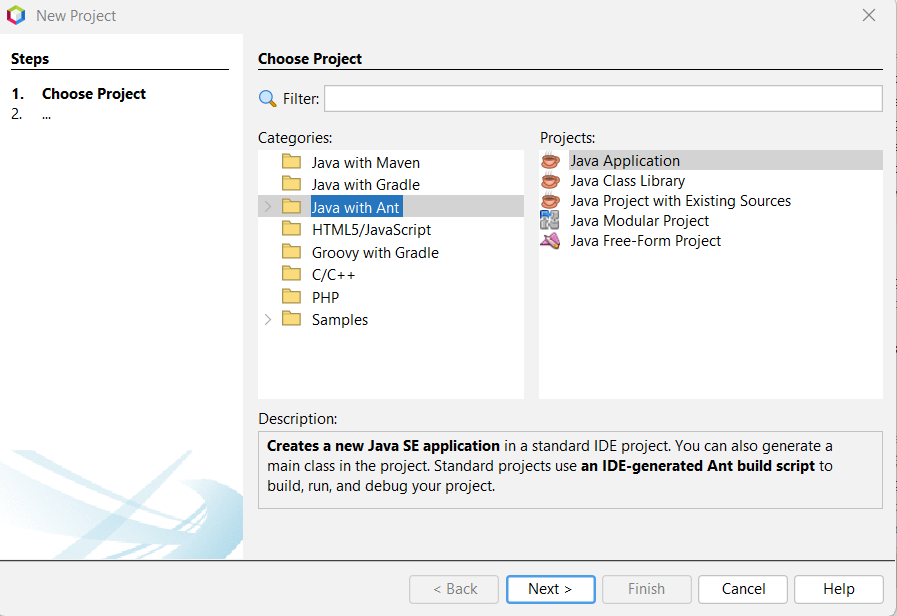


Figure 3: Ant Setup (1)

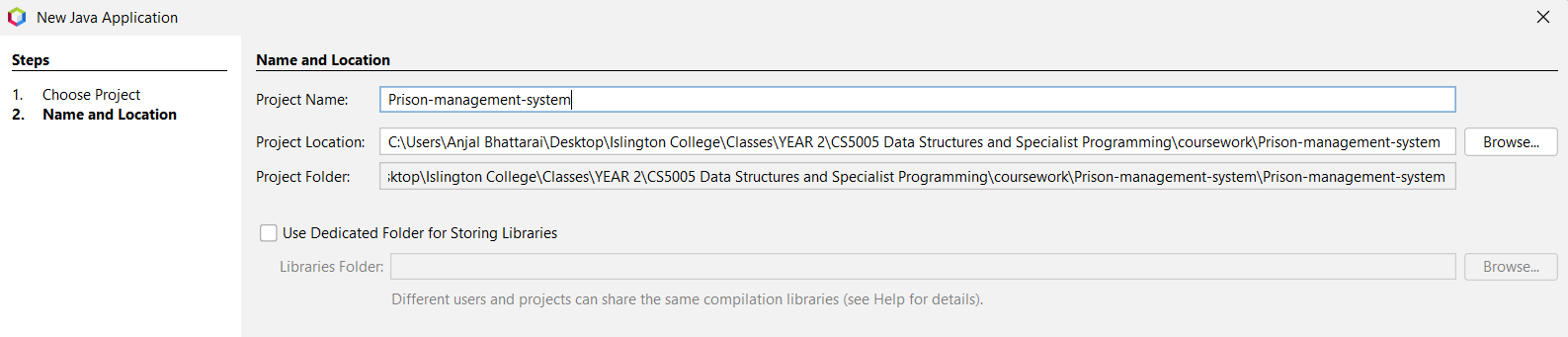


Figure 4: Ant Setup (2)

# Class Diagrams

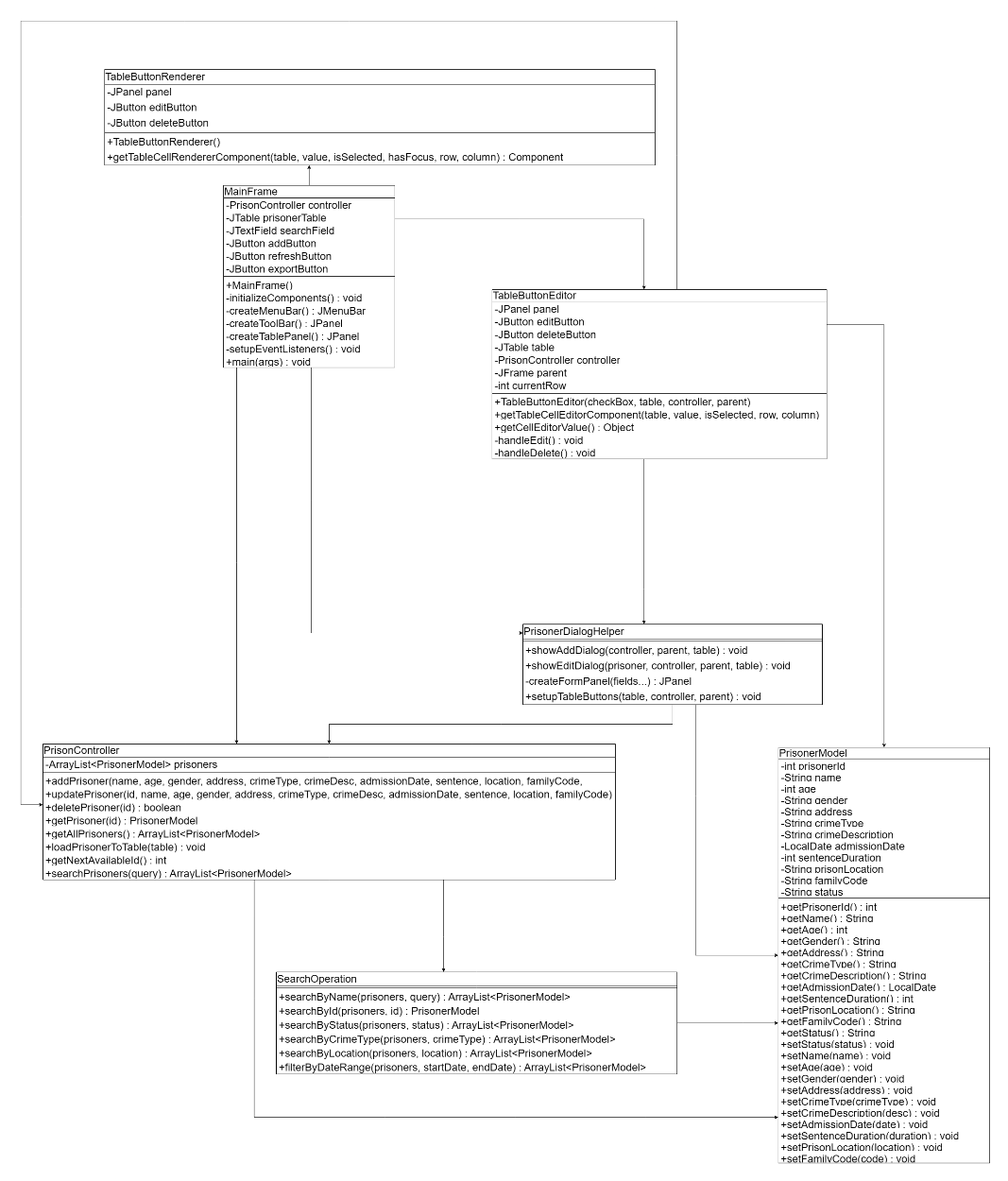
A class diagram is a structural diagram in the Unified Modeling Language (UML) that provides a static view of a system's architecture by illustrating the system's classes, their attributes, methods, and the relationships between objects. It serves as the backbone of object-oriented modeling and design, depicting how different classes interact with each other through associations, dependencies, aggregations, and compositions. Class diagrams are essential for visualizing the overall structure of a software system and act as a blueprint during development and maintenance (IBM, 2021).

Figure 5: Class diagram

In our Prison Management System, the class diagram illustrates a well-structured MVC (Model-View-Controller) architecture consisting of seven primary classes working cohesively to manage prisoner data. The PrisonerModel class serves as the data entity, encapsulating all prisoner-related attributes such as ID, name, age, crime details, and status. The PrisonController class acts as the business logic layer, managing an ArrayList of PrisonerModel objects and providing methods for CRUD operations (Create, Read, Update, Delete) and search functionality. The view layer comprises MainFrame as the primary user interface, PrisonerDialogHelper for handling add and edit dialogs with comprehensive input validation, and TableButtonRenderer and TableButtonEditor for rendering and managing action buttons within the prisoner table. Additionally, the SearchOperation class provides advanced search and filtering capabilities. The relationships demonstrate a clear separation of concerns where MainFrame depends on PrisonController for data operations, utilizes PrisonerDialogHelper for dialog management, and employs custom table components for interactive functionality, ensuring a maintainable and scalable codebase that follows object-oriented design principles.

# Wireframe Design and User Interface Screens

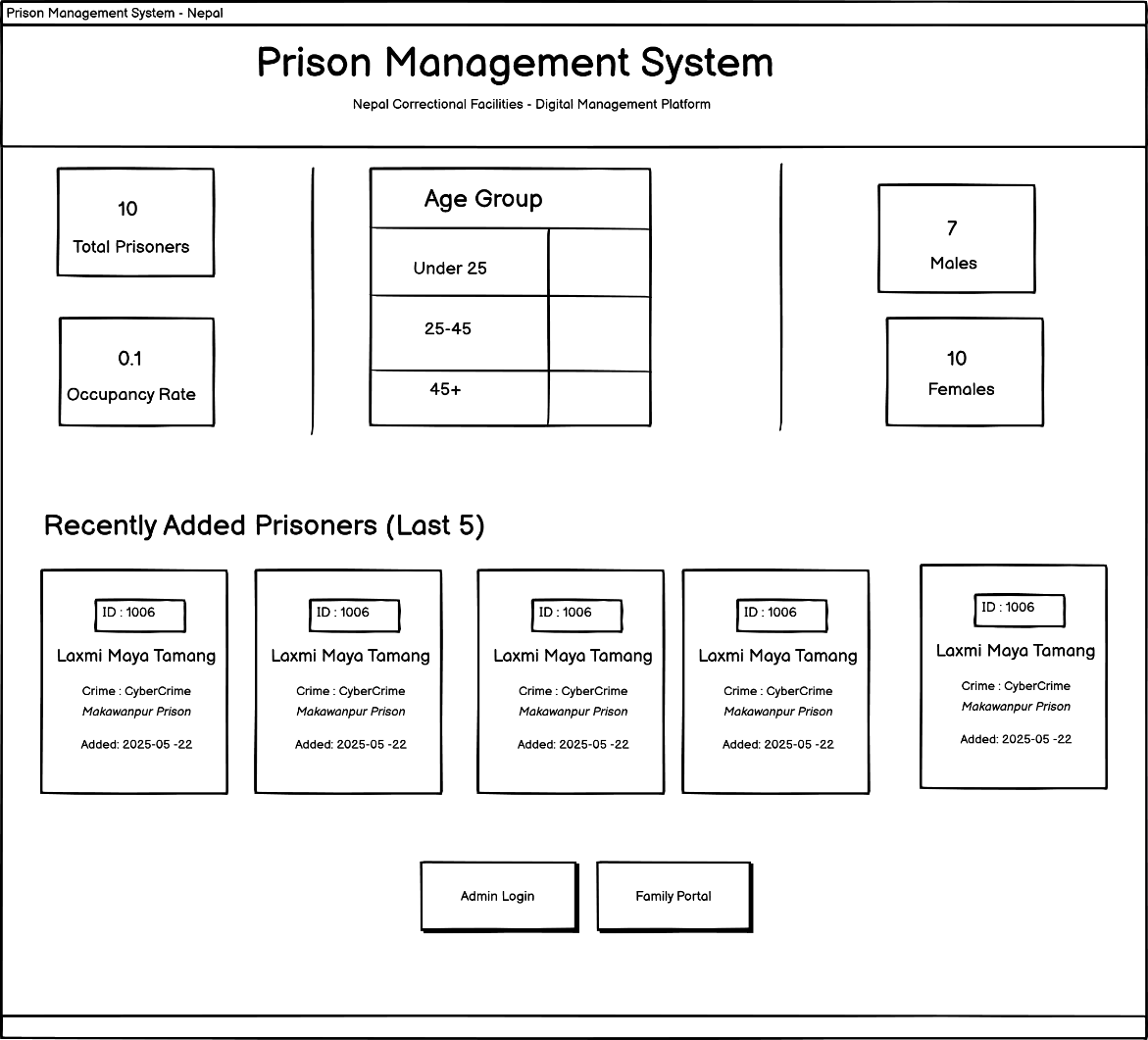


Figure 6: Wireframe of Home page

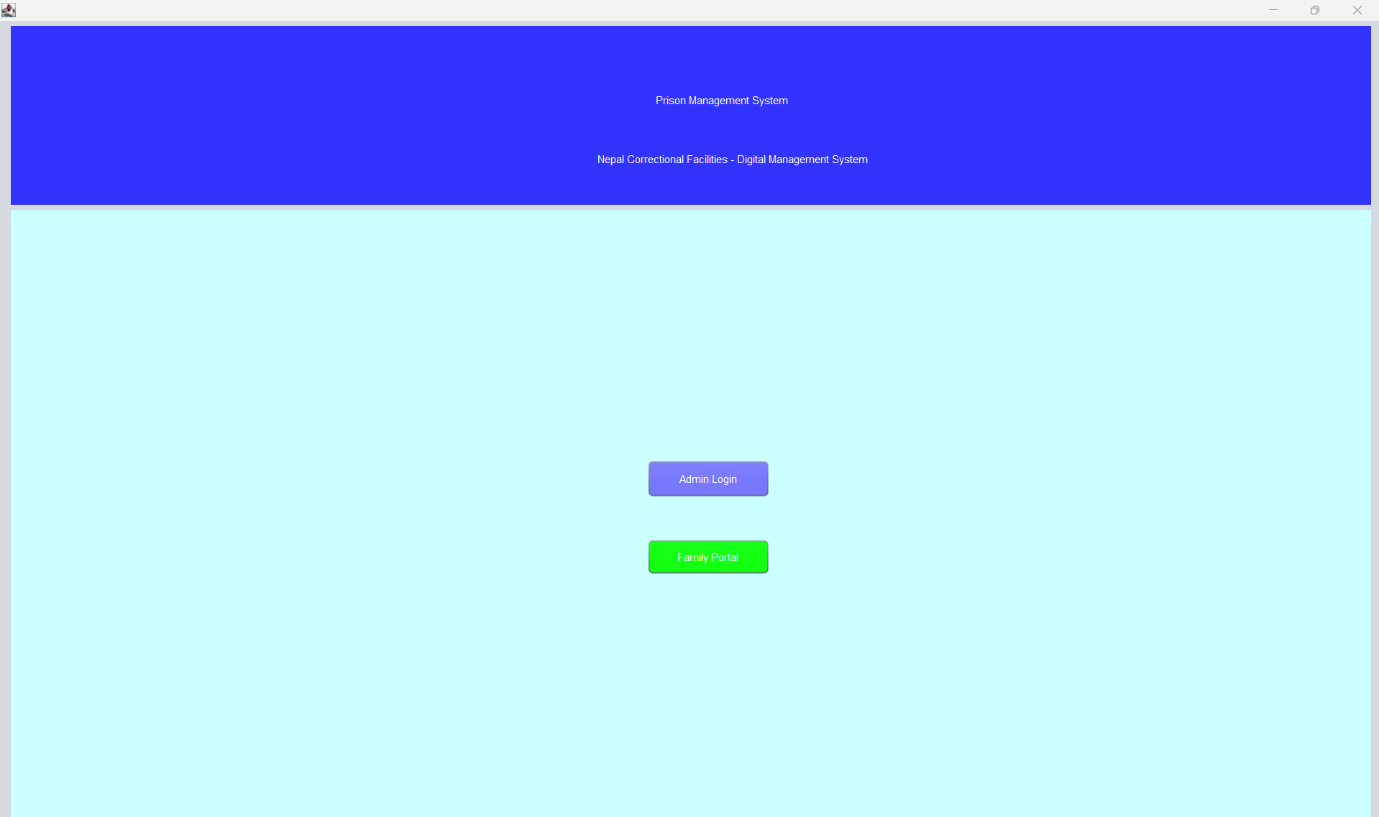


Figure 7: Actual Home page

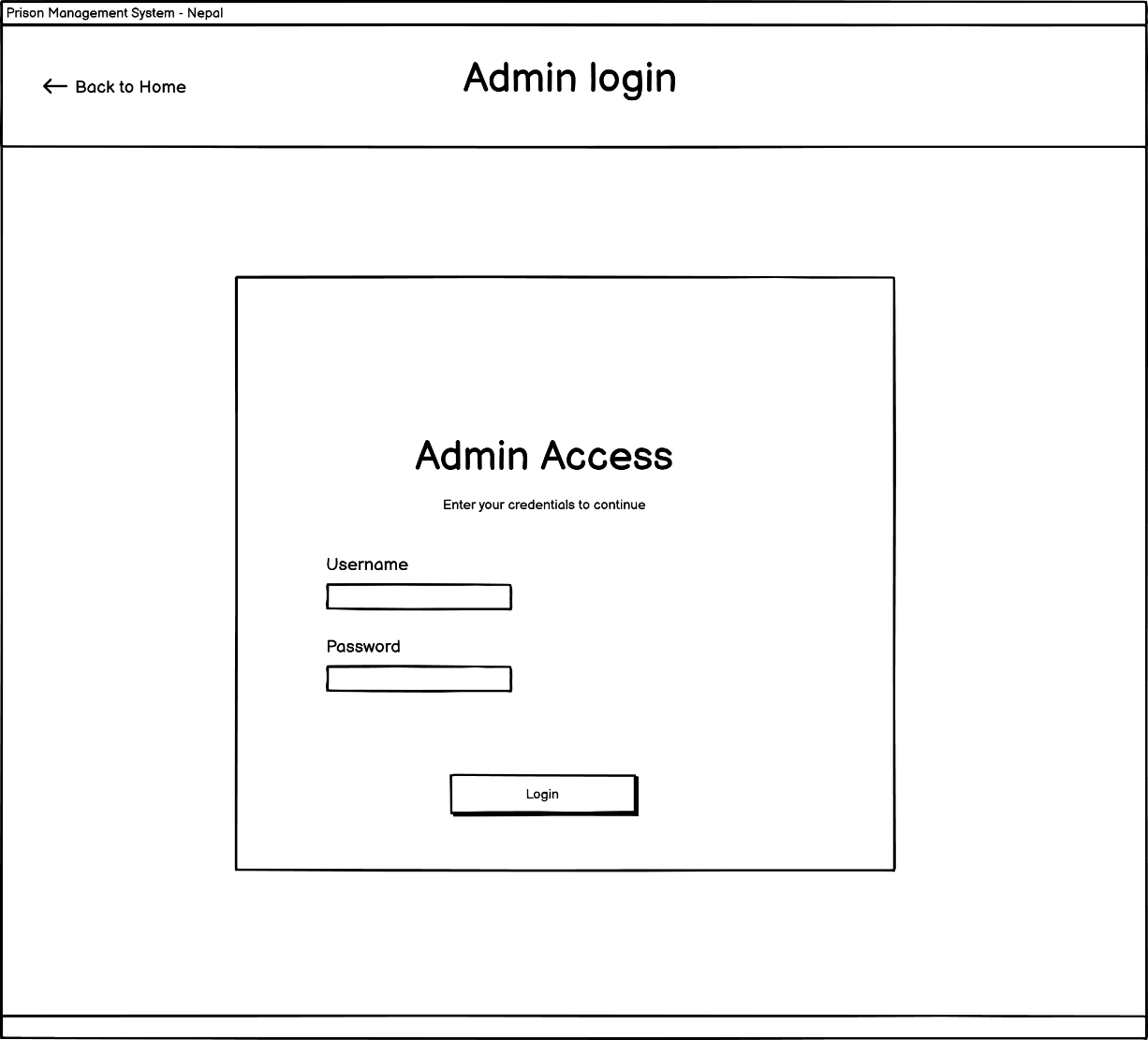


Figure 8: Wireframe of Admin Login page

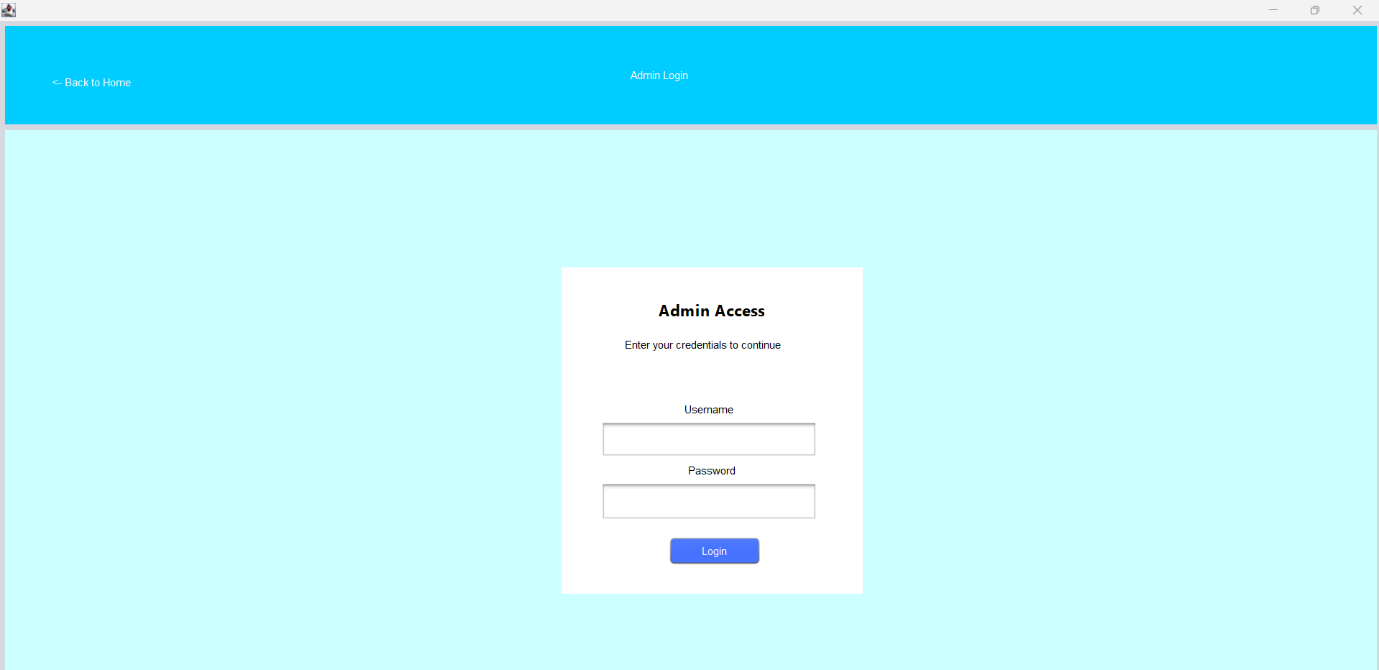


Figure 9: Actual Admin login page

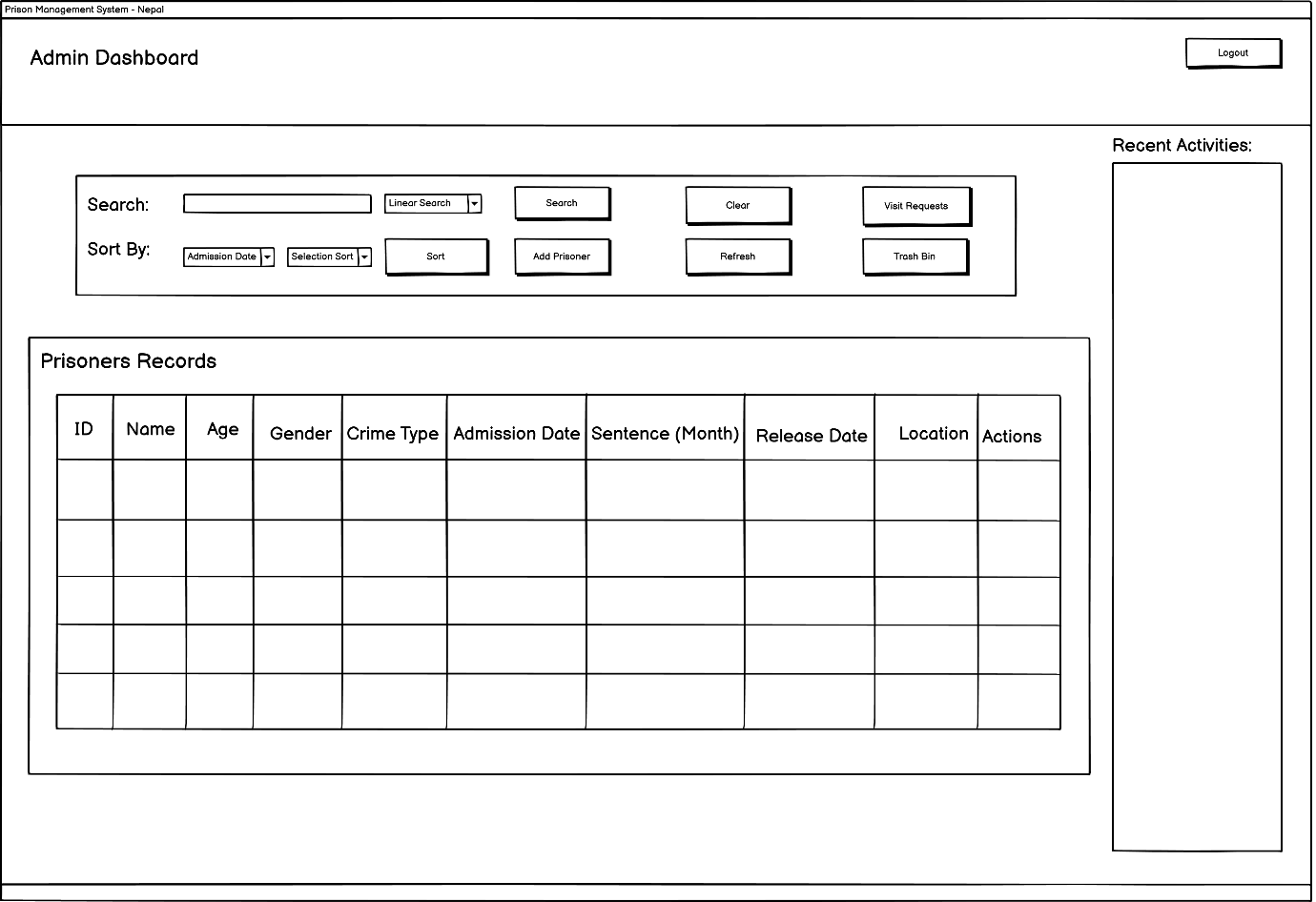


Figure 10: Wireframe of Admin Dashboard

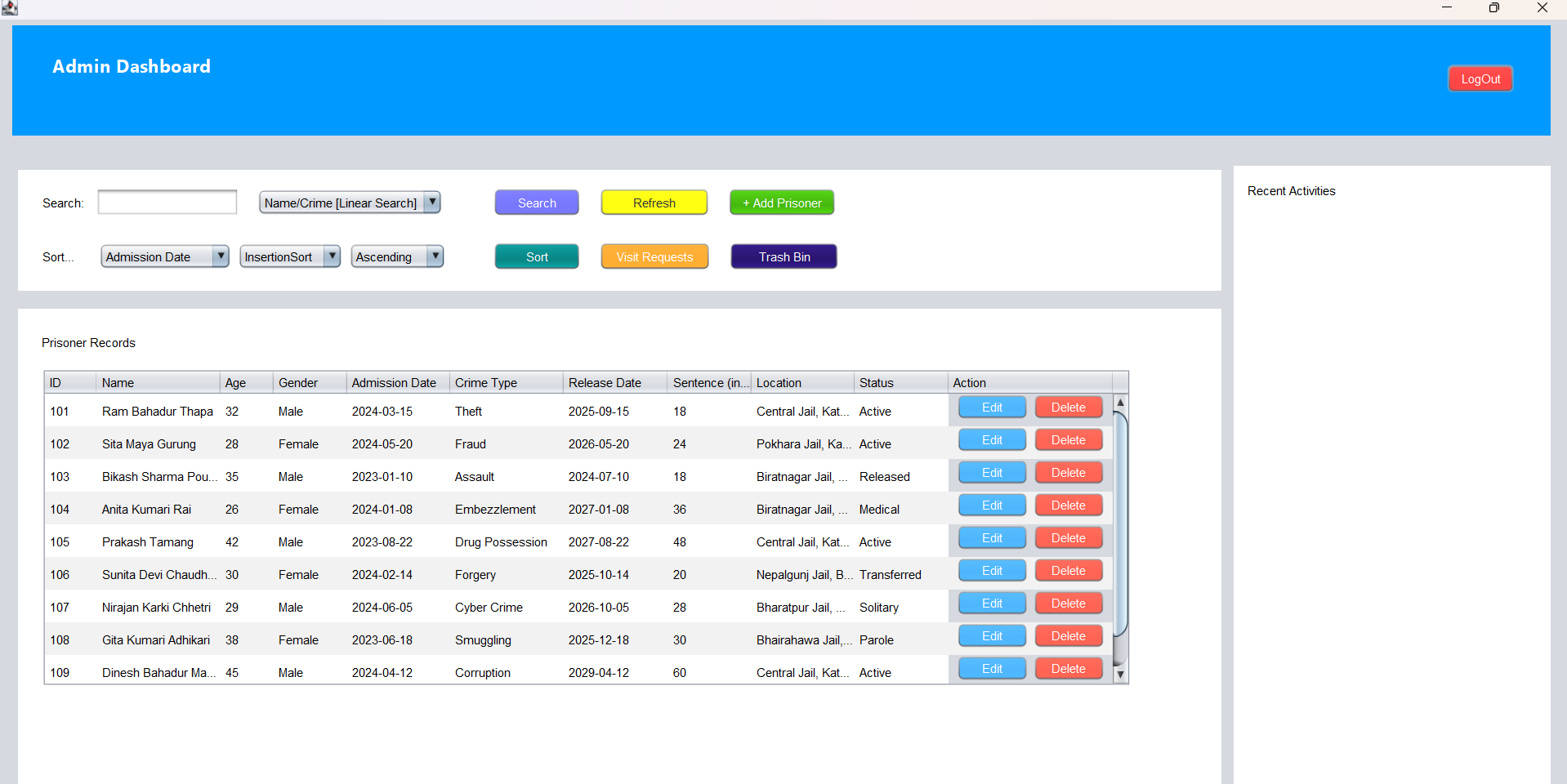


Figure : Actual Admin Login Dashboard

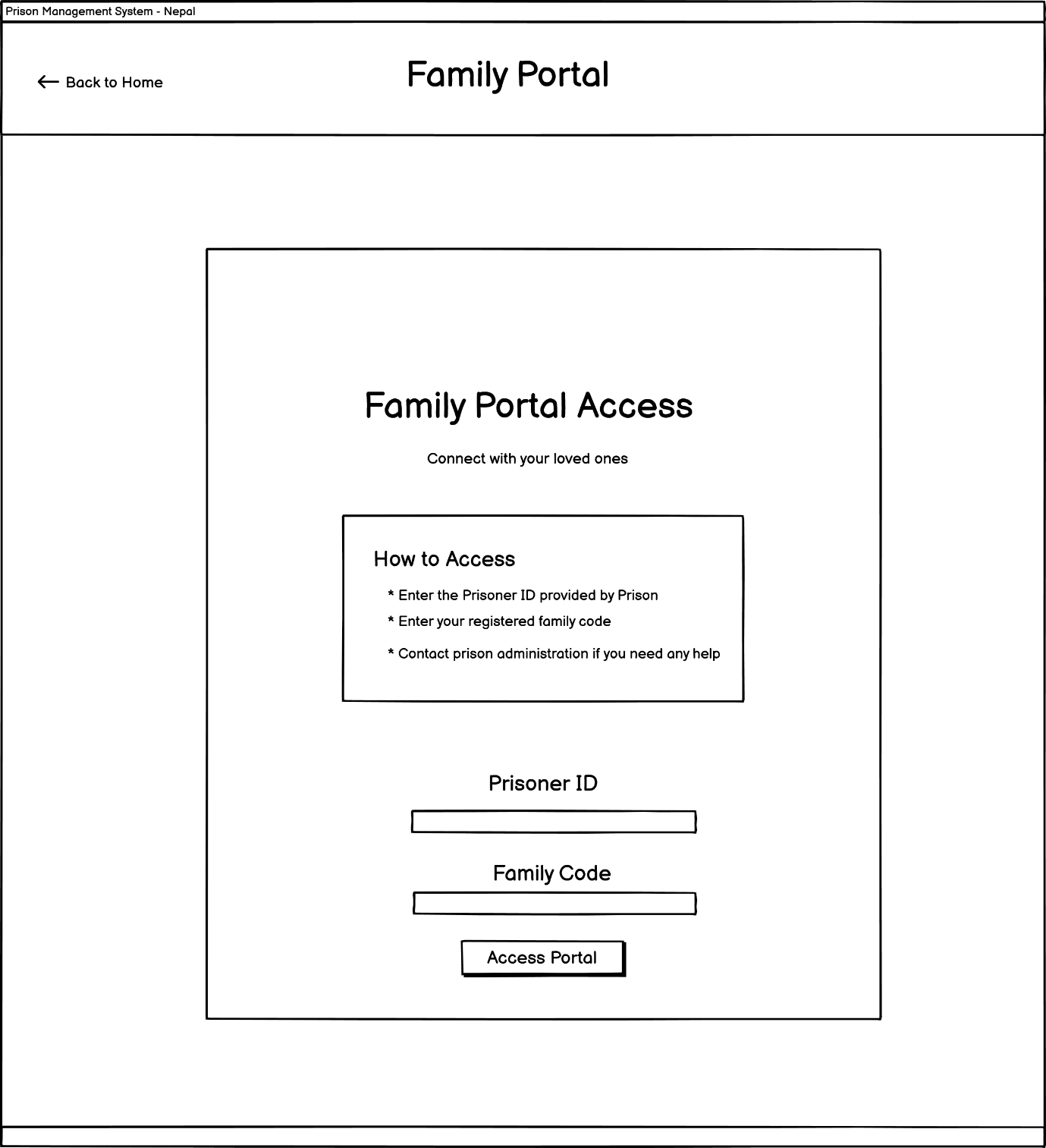


Figure 12: Wireframe of Family login portal

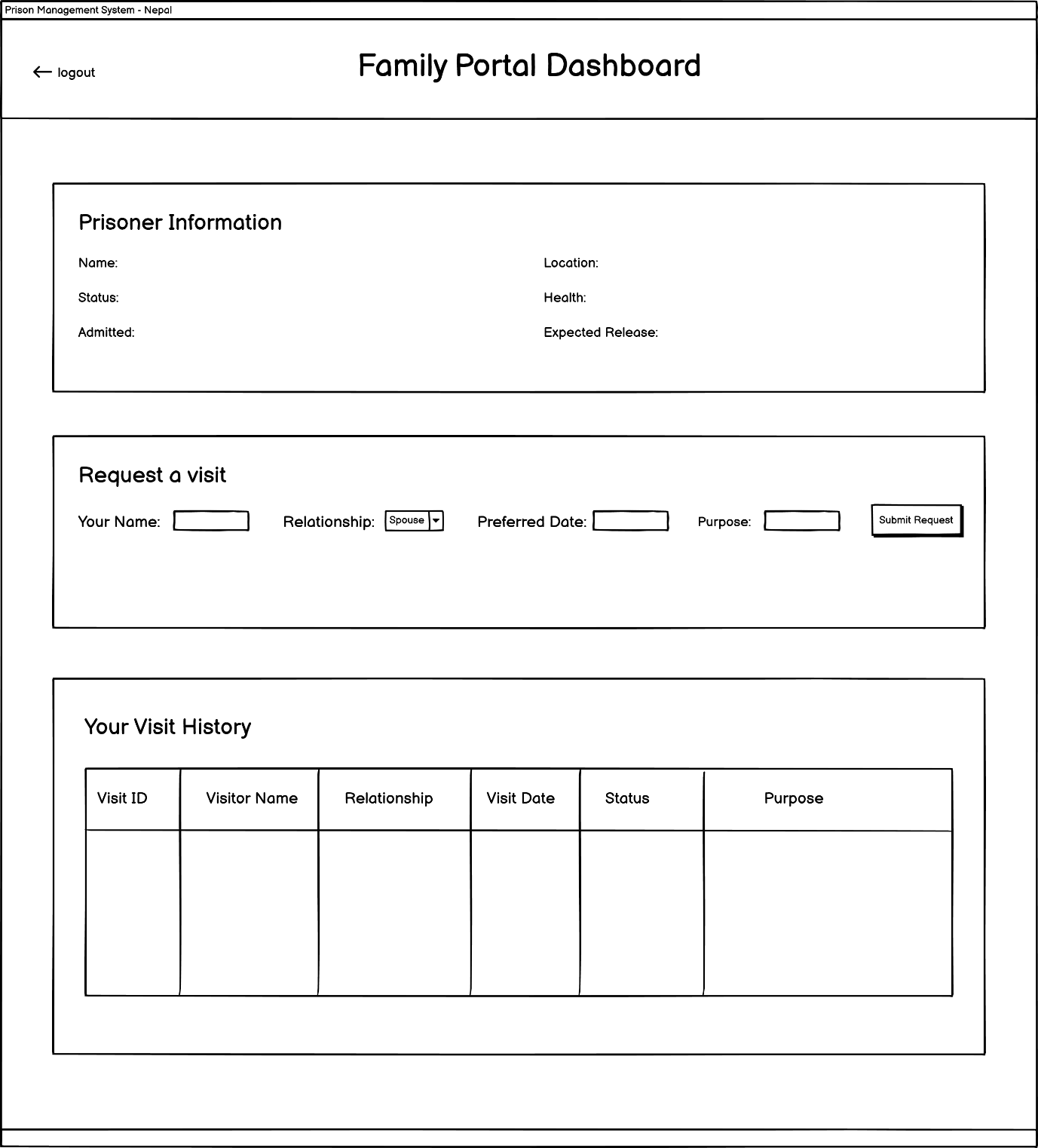


Figure 13: Wireframe of Family Dashboardp

# CRUD Functionality

## 5.1 Create Functionality

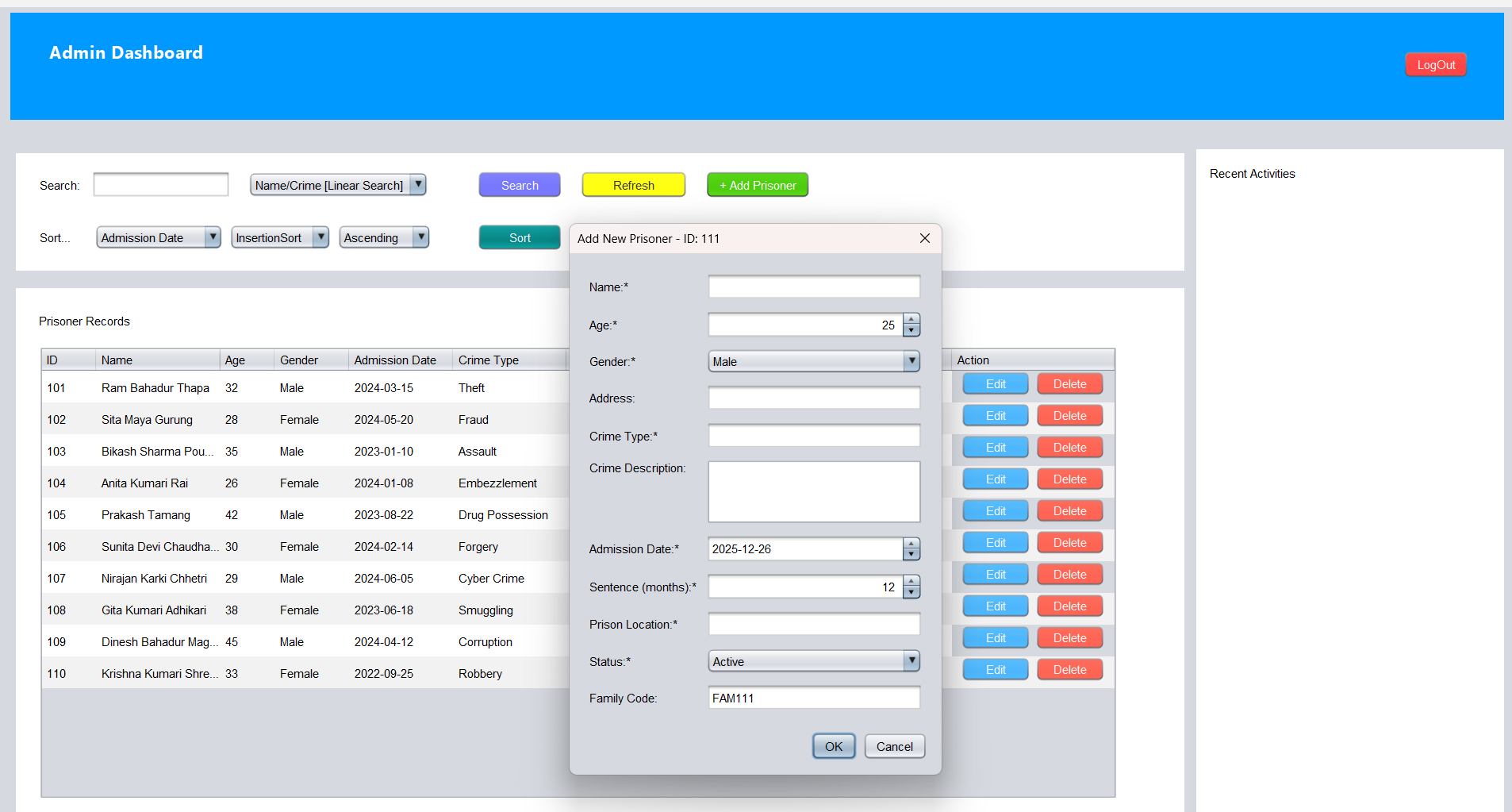


Figure 14: Dialog Box to provide new prisoner detials

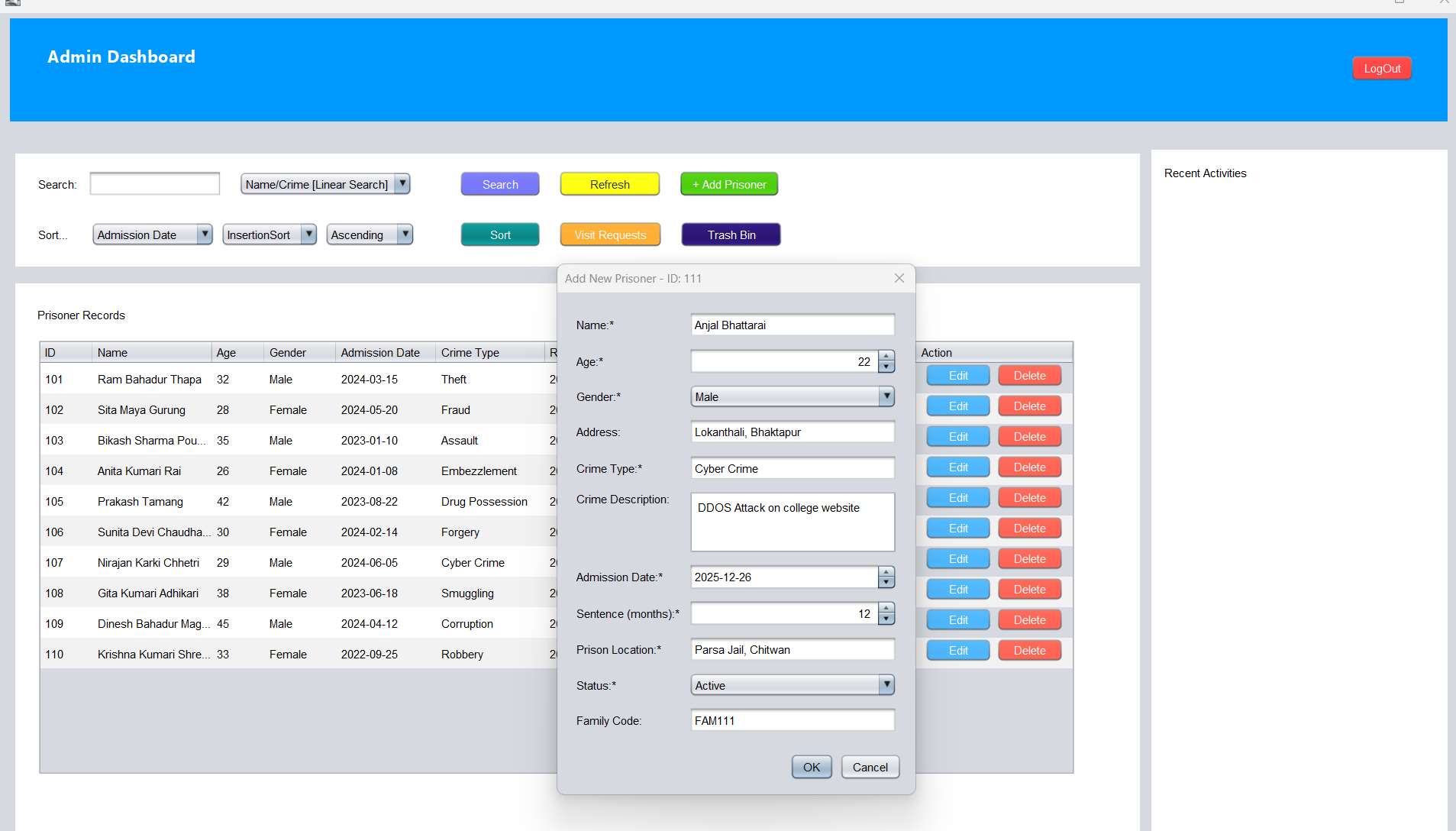


Figure 15: New Prisoner being added

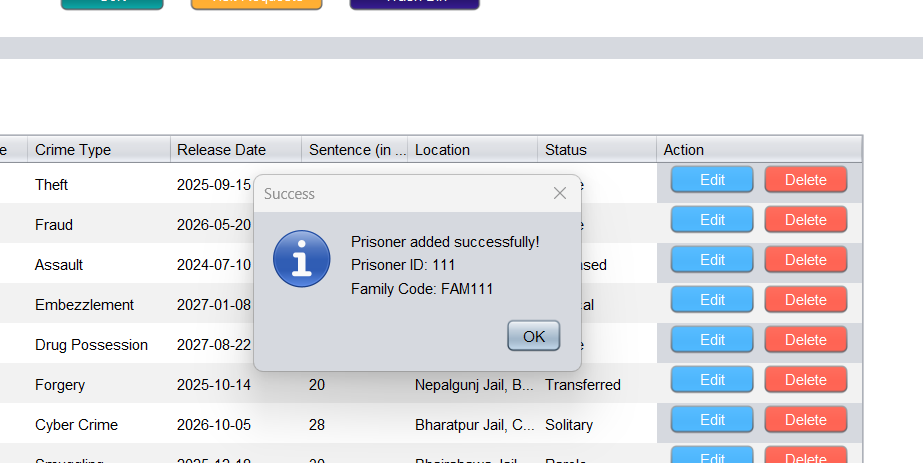


Figure 16: Create new Prisoner Successfully

## 5.2 Read Functionality

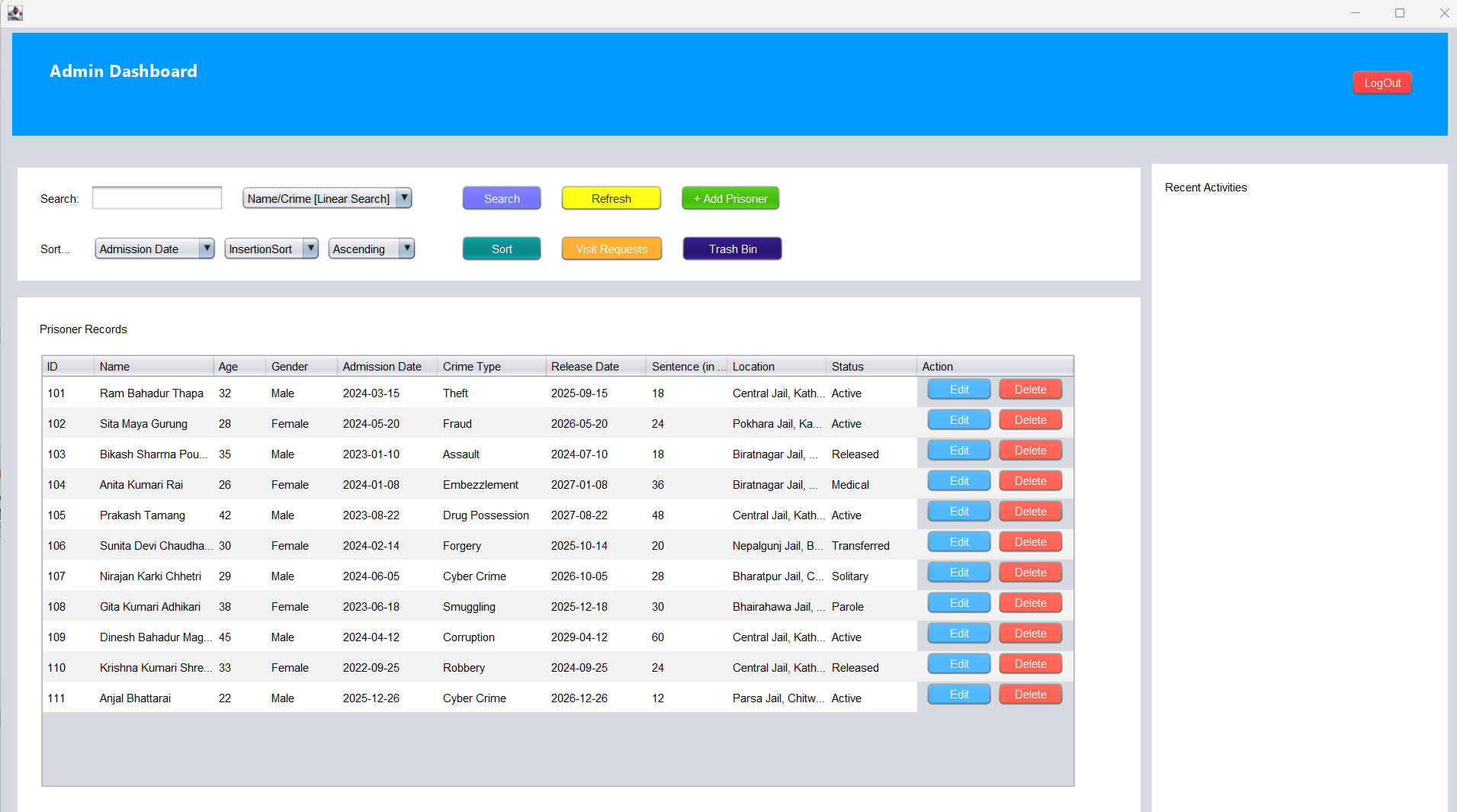


Figure 17: Detail over view of all Prisoner

## 5.3 Update Functionality

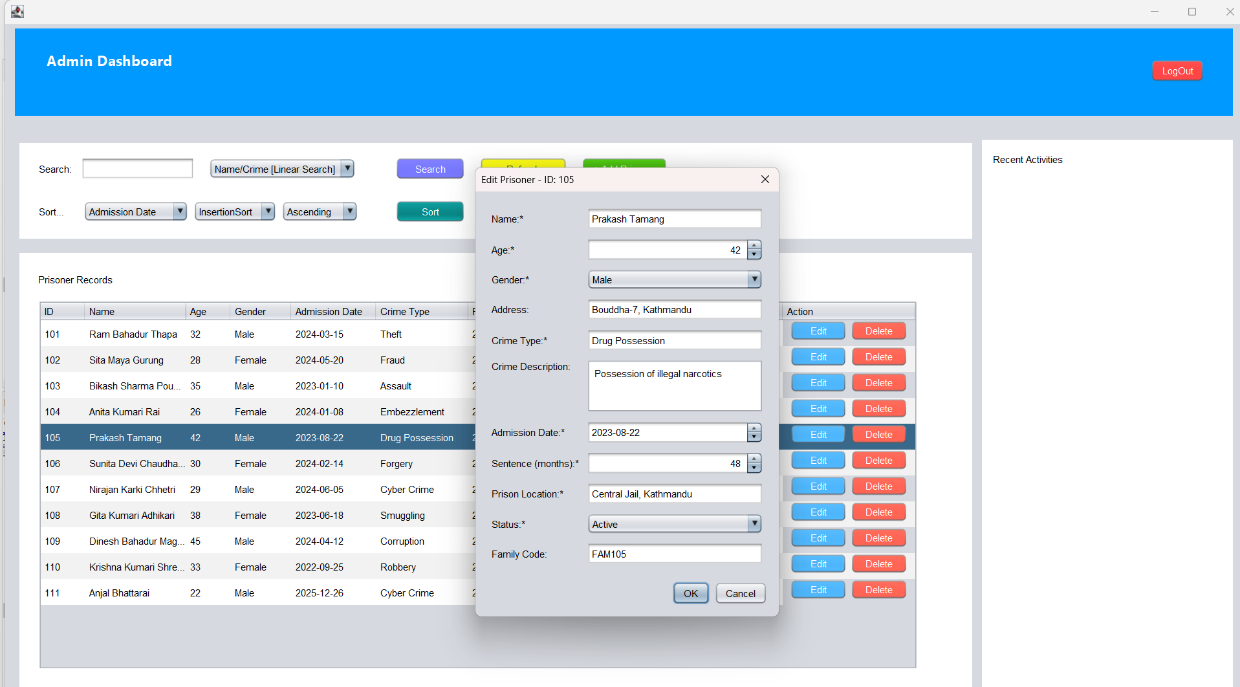


Figure 18: Editable current details of prisoner when clicked edit button

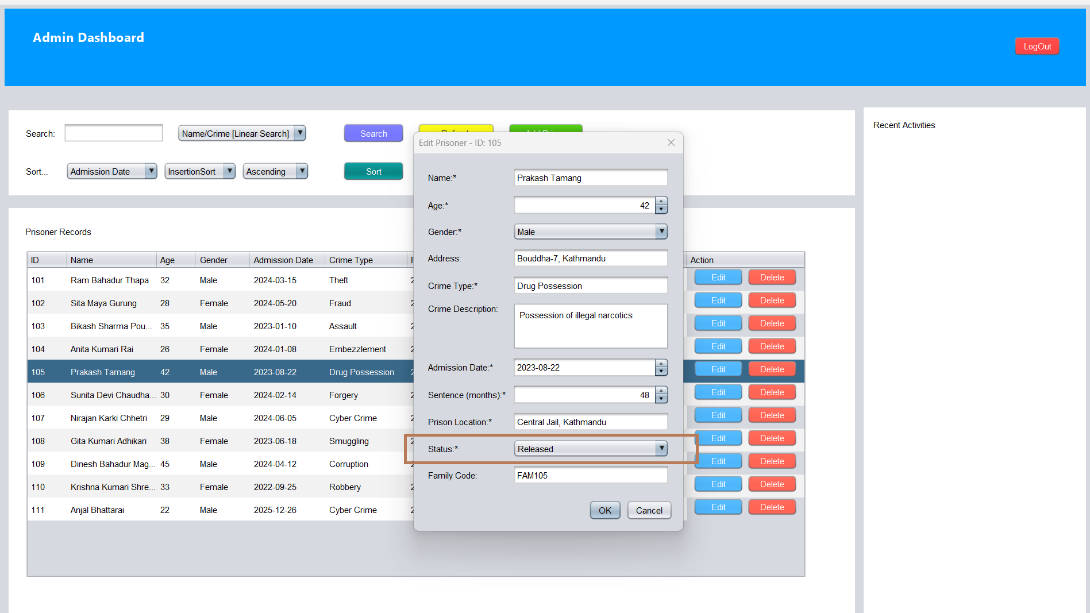


Figure 19: Status changed from active to released

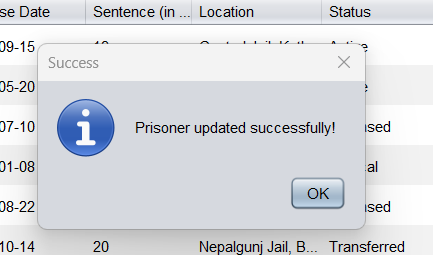




Figure 20: Updated details of prisoners

## 5.4 Delete Functionality

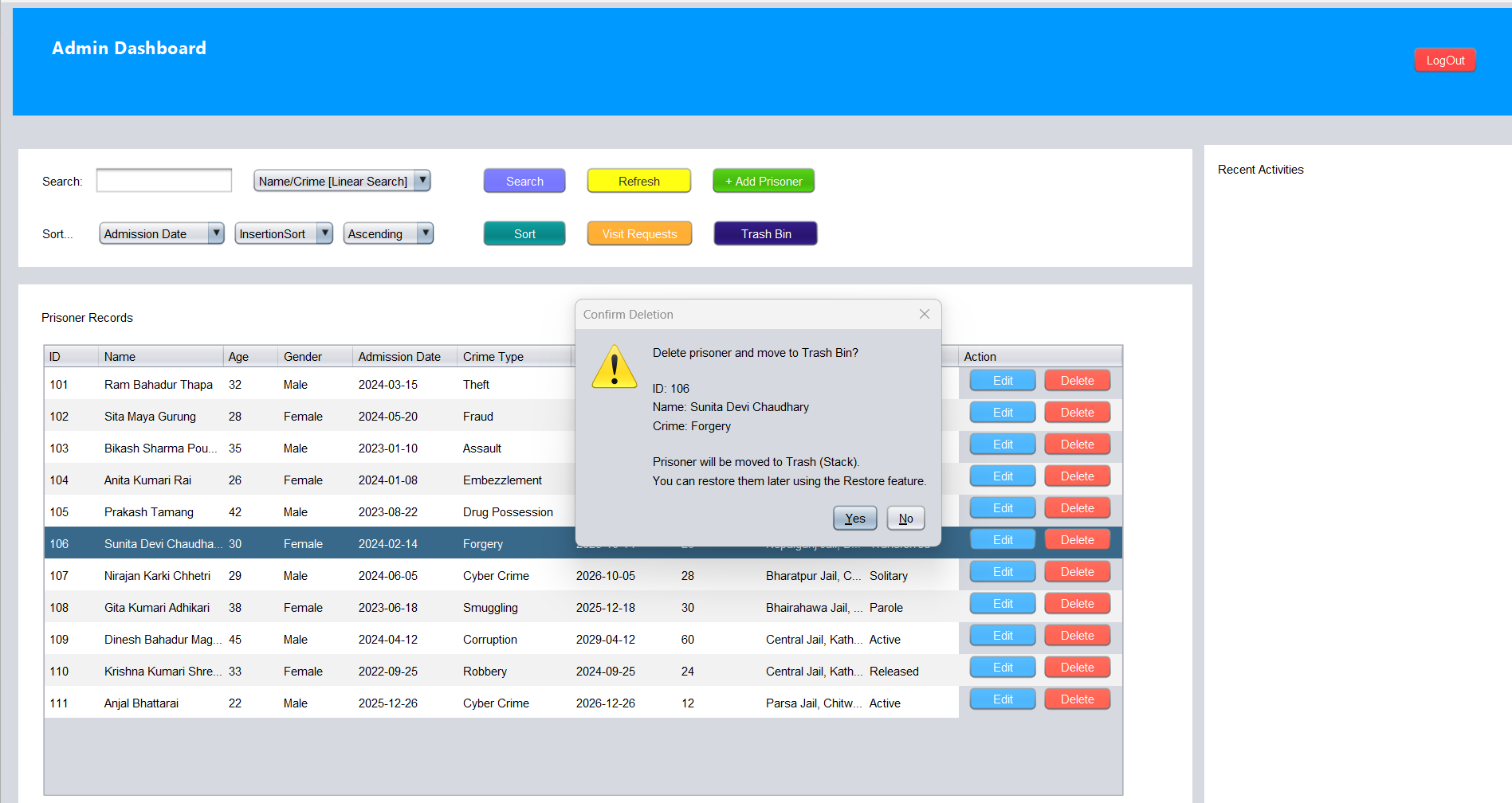


Figure 21: DIalog Box to confirm deletion of prisoner details

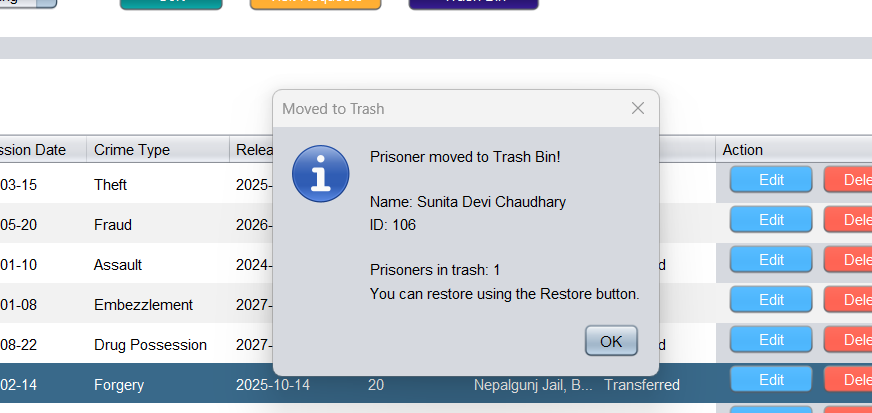


Figure 22: Deletion Confirmation Pop-up

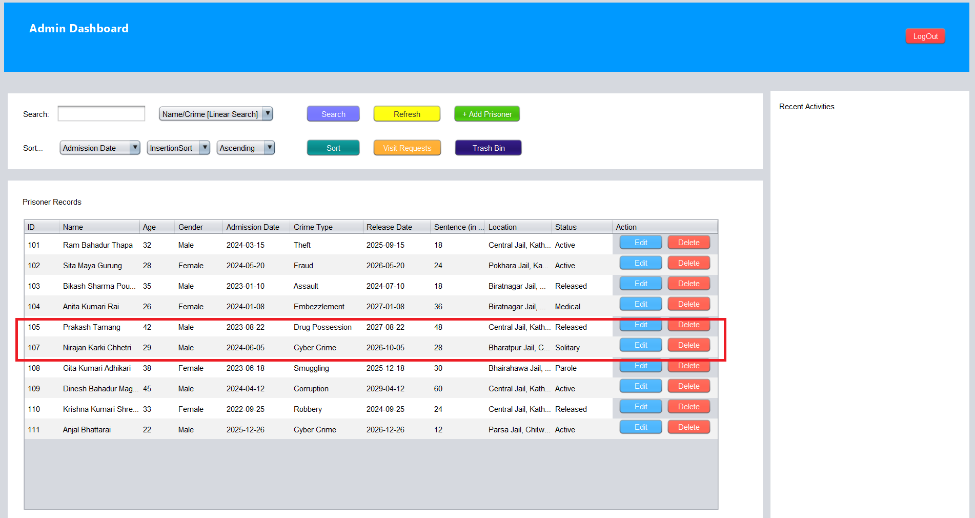


Figure 23: Prisoner with id 106 deleted

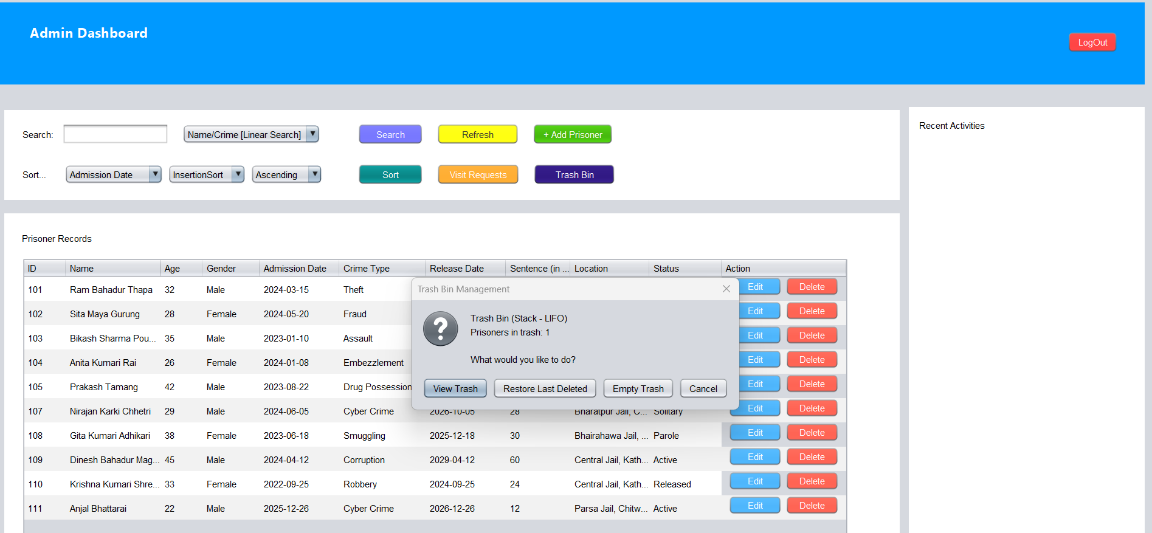


Figure 24: Recently deleted prisoner stored in STACK

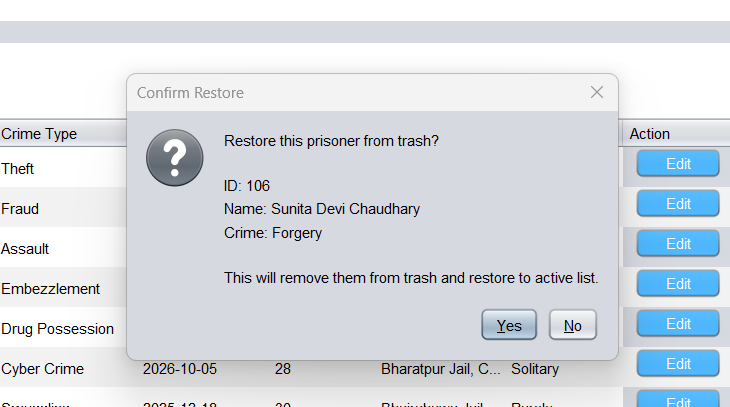


Figure 25: Restore Confirmation

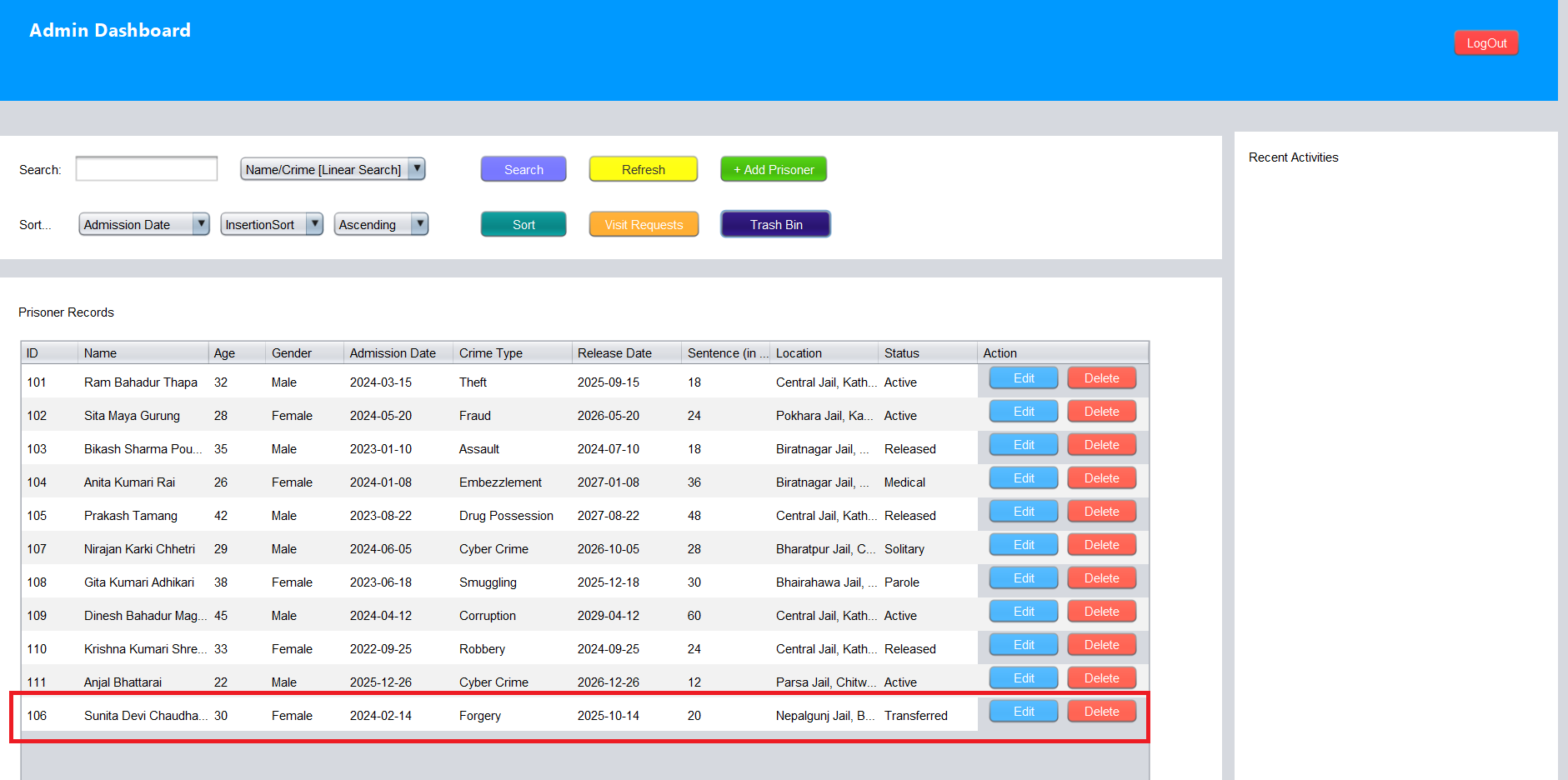


Figure 26: Restored Deleted Prisoner and Stored back in QUEUE

# Test Cases

## Test Case 01 : TC\_CREATE\_PRISONER\_EMPTY\_NAME\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 01 |
| **Test Description** | Validation: Create Prisoner with Empty Name Field |
| **Activity** | Input: Click "Add Prisoner" button, Name: "" (empty string), Age: 25, Gender: "Male", Crime: "Theft", Admission Date: "2025-12-26", Sentence Duration: 12 months, Location: "Dharan Prison", Status: "Active" |
| **Expected Output** | System should reject the entry and display error message: "Name is required and cannot be empty.” No prisoner record should be added to the LinkedList. Table should remain unchanged. |
| **Actual Output** | System rejected the entry and displayed error message: "Name is required and cannot be empty.” No prisoner record was added to the LinkedList. Table remained unchanged. |
| **Evidence** | Figure 27: Add new prisoner with empty name    Figure 28: Empty name error message displayed |
| **Remarks** | Test Successful. |

## Test Case 02: TC\_CREATE\_PRISONER\_NEGATIVE\_AGE\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 02 |
| **Test Description** | Validation: Create Prisoner with Negative Age |
| **Activity** | Input: Click "Add Prisoner" button, Name: "Phul Maya Chaudhary", Age: -5, Gender: "Female", Crime: "Assault", Admission Date: "2025-12-26", Sentence Duration: 24 months, Location: "Purwanchal Prison", Status: "Active" |
| **Expected Output** | System should reject the entry and display error message: "Age must atleast 18 years. No prisoner record should be created. PrisonerRecordTable should not update. |
| **Actual Output** | System rejected the entry and displayed error message: "Age must atleast 18 years. No prisoner record was created. PrisonerRecordTable not updated. |
| **Evidence** | Figure 29: Error message for negative age |
| **Remarks** | Test Successful |

## Test Case 03: CREATE\_PRISONER\_NEGATIVE\_SENTENCE\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 03 |
| **Test Description** | Validation: Create Prisoner with Negative Sentence Duration |
| **Activity** | Input: Click "Add Prisoner" button, Name: "Jane Smith", Age: 30, Gender: "Female", Crime: "Fraud", Admission Date: "15/12/2024", Sentence Duration: -10 months, Location: "Cell Block C", Status: "Active" |
| **Expected Output** | System should handle validation and display error message: "Sentence duration must be atleast 1 month." Record should not be added to LinkedList. |
| **Actual Output** | System handled validation and displayed error message: "Sentence duration must be atleast 1 month." Record was not added to LinkedList. |
| **Evidence** | Figure : Error message for negative sentence period |
| **Remarks** | Test Successful |

## Test Case 04: CREATE\_PRISONER\_INVALID\_DATE\_FORMAT\_EXCEPTION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 04 |
| **Test Description** | Exception Handling: Create Prisoner with Invalid Date Format |
| **Activity** | Input: Click "Add Prisoner" button, Name: "Phul Maya Tamang", Age: 28, Gender: "Female", Crime: "Burglary", Admission Date: "invalid-date" or "32/13/2025", Sentence Duration: 18 months, Location: "Kathmandu", Status: "Active" |
| **Expected Output** | System should handle DateTimeParseException gracefully and display error message: "Invalid date format. Please use DD/MM/YYYY". No record should be created. |
| **Actual Output** | System handled DateTimeParseException gracefully and displayed error message: "Invalid date format. Please use DD/MM/YYYY" . No record was created. |
| **Evidence** | Figure 31: Add Prisoner with invalide Date format    Figure 32: Invalid Date Format Error message |
| **Remarks** | Test Successful |

**Test Case 05**

**Name:** TC\_READ\_SEARCH\_NON\_NUMERIC\_ID\_EXCEPTION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 05 |
| **Test Description** | Exception Handling: Read/Search with Non-Numeric ID (Binary Search) |
| **Activity** | Input: Select Search Type: "ID Binary Search", Enter Search Term: "ABC123" (String instead of number), Click "Search" button |
| **Expected Output** | System should handle NumberFormatException and display error message: "Please enter a valid numeric ID" or "ID must be a number". Search should not execute. No system crash should occur. Console should show exception handling message. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshot showing error dialog + Console output] |

**Test Case 06**

**Name:** TC\_READ\_SEARCH\_EMPTY\_SEARCH\_TERM\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 06 |
| **Test Description** | Validation: Read/Search with Empty Search Term |
| **Activity** | Input: Select Search Type: "Name/Crime Linear Search", Search Term: "" (empty), Click "Search" button |
| **Expected Output** | System should display warning message: "Please enter a search term" before executing search. Search operation should not proceed. Table should remain unchanged showing all prisoners. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshot of warning dialog] |

**Test Case 07**

**Name:** TC\_UPDATE\_PRISONER\_NON\_EXISTENT\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 07 |
| **Test Description** | Validation: Update Non-Existent Prisoner Record |
| **Activity** | Input: Manually attempt to update prisoner with ID: 99999 (doesn't exist), Modified fields: Name, Age, Crime, etc. |
| **Expected Output** | System should detect non-existent prisoner and display error message: "Prisoner with ID 99999 not found" or "Cannot update: Prisoner does not exist". Update operation should be cancelled. LinkedList should remain unchanged. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshot of error message] |

**Test Case 08**

**Name:** TC\_UPDATE\_PRISONER\_INVALID\_AGE\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 08 |
| **Test Description** | Validation: Update Prisoner with Invalid Age Value |
| **Activity** | Input: Click "Edit" button on existing prisoner row, Keep all fields same except: Age: -15 or 0 or 200 (out of valid range), Click "Update" |
| **Expected Output** | System should reject update and display error message: "Age must be between 18 and 150" or "Please enter a valid age". Original prisoner data should remain unchanged in LinkedList and table. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshot showing validation error] |

**Test Case 09**

**Name:** TC\_DELETE\_NON\_EXISTENT\_PRISONER\_VALIDATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 09 |
| **Test Description** | Validation: Delete Non-Existent Prisoner Record |
| **Activity** | Input: Attempt to delete prisoner with ID: 99999 (doesn't exist in LinkedList), Click "Delete" button |
| **Expected Output** | System should display error message: "Prisoner with ID 99999 not found" or "Cannot delete: Prisoner does not exist". No deletion should occur. Trash bin (Stack) should not be updated. Table should remain unchanged. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshot of error dialog] |

**Test Case 10**

**Name:** TC\_DELETE\_AND\_RESTORE\_STACK\_OPERATION

| **Field** | **Details** |
| --- | --- |
| **Test Case** | 10 |
| **Test Description** | Functional Testing: Delete Prisoner and Restore from Trash (Stack LIFO) |
| **Activity** | Input: Select an existing prisoner (e.g., ID: 101), Click "Delete" button (Prisoner moved to trash - Stack push), Click "Trash Bin" button, Select "Restore Last Deleted" (Stack pop operation), Click "Refresh" to verify |
| **Expected Output** | Delete: Prisoner removed from LinkedList and pushed to trash Stack. Success message: "Prisoner deleted and moved to trash". Restore: Last deleted prisoner popped from Stack and restored to LinkedList. Success message: "Prisoner restored successfully". Prisoner should reappear in PrisonerRecordTable with same data. Console should show Stack LIFO operations. |
| **Actual Output** | [To be filled after testing] |
| **Evidence** | [Screenshots: Before delete, After delete, Trash contents, After restore + Console output] |

# Conclusion

The Prison Management System project effectively demonstrates how core concepts of data structures and algorithms can be applied to a realistic and socially relevant administrative domain. By contextualizing theoretical computer science principles within the challenges faced by Nepal’s overcrowded prison system, the project successfully bridges the gap between abstract learning and practical implementation. The adoption of the Model-View-Controller (MVC) architecture ensures a clear separation of concerns, enabling organized code structure, improved maintainability, and scalability. Through distinct responsibilities assigned to the model, controller, and view components, the system reflects sound software engineering practices aligned with industry standards.

The implementation of fundamental data structures such as Linked Lists, Queues, and Stacks addresses the dynamic nature of prison data, including frequent admissions, releases, and record updates. Additionally, the use of searching and sorting algorithms provides efficient data retrieval and structured presentation, reinforcing algorithmic reasoning within a real-world dataset. These design choices not only satisfy academic requirements of the CS5005 module but also illustrate how disciplined application of algorithms can enhance administrative efficiency, transparency, and data consistency.

Beyond its role as an educational exercise, this project offers conceptual insight into how information systems can support institutional decision-making in complex environments. Features such as input validation, modular design, and a user-friendly interface emphasize the importance of reliability and usability in administrative software. Overall, the Prison Management System reinforces the value of algorithmic thinking, structured design, and architectural planning in developing scalable and maintainable systems, while preparing students with practical competencies applicable to future professional and academic endeavors.

# Future Works

While the Prison Management System fulfills its academic objectives and demonstrates effective application of data structures and software design principles, several enhancements can be explored in future iterations to improve scalability, functionality, and real-world applicability.

1. **Advanced Data Structures and Algorithms**

Future versions of the system may incorporate more advanced data structures to improve efficiency and analytical capability. Hash Tables could be used to enable faster prisoner record retrieval in large datasets, while Priority Queues may support handling of urgent cases such as medical emergencies or court hearings. Tree-based structures, such as balanced search trees, could maintain sorted records efficiently and support statistical queries for reporting and analysis.

1. **Database Integration and Persistence**

Currently, the system operates with in-memory data storage, limiting persistence and scalability. Integrating a relational database would allow permanent data storage, concurrent access, and structured querying using SQL. This enhancement would also support features such as data backup, audit logging, and multi-user access, making the system more suitable for institutional use.

1. **Web-Based Deployment and API Support**

Migrating the application from a desktop-based Java Swing environment to a web-based architecture would improve accessibility and deployment flexibility. A web application with RESTful APIs would allow integration with external systems such as court or law enforcement platforms and enable access across multiple devices without local installation.

1. **Security and Access Control**

Future development should strengthen system security through role-based access control to ensure that sensitive data is accessed only by authorized personnel. Implementing authentication mechanisms, encrypted data handling, and activity logging would improve data protection and accountability.

1. **Basic Analytics and Reporting**

The system could be extended with basic reporting features, such as occupancy summaries, crime-type distributions, and release forecasts. Simple visualizations and downloadable reports would support administrative decision-making without introducing unnecessary system complexity.

# Bibliography

Anon., n.d. *MVC Framework - Introduction.* [Online]   
Available at: https://www.tutorialspoint.com/mvc\_framework/mvc\_framework\_introduction.htm  
[Accessed 26 December 2025].

Apache Ant Project, n.d. *Apache Ant Manual.* [Online]   
Available at: https://ant.apache.org/manual/  
[Accessed 26 December 2025].

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1994 . Design Patterns: Elements of Reusable Object-Oriented Software. In: s.l.:Addison-Wesley.

GeeksforGeeks, 2023. *MVC Architecture.* [Online]   
Available at: https://www.geeksforgeeks.org/mvc-design-pattern/  
[Accessed 26 December 2025].

IBM, 2021. *UML Basics: The Class Diagram.* [Online]   
Available at: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-class  
[Accessed 26 December 2025].

Kathmandu School of Law Review, 2022. *Prison Management and Overcrowding in Nepal.* [Online]   
Available at: https://kslreview.org/index.php/kslr/article/download/2201/1541  
[Accessed 2026 December 2025].

The Kathmandu Post, 2023. *Violent clashes continue in Nepal’s overcrowded prisons.* [Online]   
Available at: https://kathmandupost.com/national/violent-clashes-continue-in-nepal-s-overcrowded-prisons  
[Accessed 26 December 2025].

# 10. Appendix