



**Module Code & Module Title**

**CS5003NI Data Structures and Specialist Programming**

**30% Individual Coursework 1**

**Submission: Final Submission**.

**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: 16/01/2025**

**Assignment Submission Date: 16/01/2025**

**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](#_Toc219302988)

[1.1 Pupose 3](#_Toc219302989)

[1.2 Audience 3](#_Toc219302990)

[1.3 Aim and Objectives 4](#_Toc219302991)

[1.4 Problem Statement 5](#_Toc219302992)

[2 MVC Architecture & ANT Setup 6](#_Toc219302993)

[3 Class Descriptions 9](#_Toc219302994)

[3.1 Model Package 10](#_Toc219302995)

[3.1.1 Prisoner Model 11](#_Toc219302996)

[3.1.2 Visit Request Model 14](#_Toc219302997)

[3.1.3 Activity Model 17](#_Toc219302998)

[3.2 View Package 19](#_Toc219302999)

[3.2.1 MainFrame 20](#_Toc219303000)

[3.2.2 Prisoner Dialog Helper 24](#_Toc219303001)

[3.2.3 Table Button Editor 26](#_Toc219303002)

[3.2.4 Table Button Renderer 28](#_Toc219303003)

[3.2.5 Trash Bin Dialog 29](#_Toc219303004)

[3.2.6 View Details Dialog 31](#_Toc219303005)

[3.2.7 Visit Requests Dialog 33](#_Toc219303006)

[3.3 Controller Package 37](#_Toc219303007)

[3.3.1 CRUD 38](#_Toc219303008)

[3.3.2 OperationResult 41](#_Toc219303009)

[3.3.3 PrisonController 44](#_Toc219303010)

[3.3.5 SimpleQueue 50](#_Toc219303011)

[3.3.6 SimpleStack 52](#_Toc219303012)

[3.3.7 SortOperation 54](#_Toc219303013)

[3.3.8 TrashBinOperation 56](#_Toc219303014)

[3 Wireframe Design and User Interface Screens 58](#_Toc219303015)

[4 CRUD Functionality 64](#_Toc219303016)

[5.1 Create Functionality 64](#_Toc219303017)

[5.2 Read Functionality 65](#_Toc219303018)

[5.3 Update Functionality 66](#_Toc219303019)

[5.4 Delete Functionality 68](#_Toc219303020)

[5 Test Cases 70](#_Toc219303021)

[Test Case 01 : TC\_CREATE\_PRISONER\_EMPTY\_NAME\_VALIDATION 70](#_Toc219303022)

[Test Case 02: TC\_CREATE\_PRISONER\_NEGATIVE\_AGE\_VALIDATION 72](#_Toc219303023)

[Test Case 03: CREATE\_PRISONER\_NEGATIVE\_SENTENCE\_VALIDATION 73](#_Toc219303024)

[Test Case 04: CREATE\_PRISONER\_INVALID\_DATE\_FORMAT\_EXCEPTION 74](#_Toc219303025)

[6 Conclusion 80](#_Toc219303026)

[7 Future Works 81](#_Toc219303027)

[8 Bibliography 83](#_Toc219303028)

[10. Appendix 84](#_Toc219303029)

**Table of Figures**

[Figure 1: MVC Architechture 7](#_Toc219309343)

[Figure 2: MVC Folder structure for Prison Management System 7](#_Toc219309344)

[Figure 3: Ant Setup (1) 8](#_Toc219309345)

[Figure 4: Ant Setup (2) 8](#_Toc219309346)

[Figure 5: Complete Class diagram 9](#_Toc219309347)

[Figure 6: Model Package class diagram 10](#_Toc219309348)

[Figure 7: Class Diagram for PrisonerModel 11](#_Toc219309349)

[Figure 8: Class Diagram for VisitRequestModel 14](#_Toc219309350)

[Figure 9: Class Diagram for ActivityModel 16](#_Toc219309351)

[Figure 10: View Package Class Diagram 18](#_Toc219309352)

[Figure 11: Controller Package Class Diagrams 36](#_Toc219309353)

[Figure 13: Wireframe of Landing page 55](#_Toc219309354)

[Figure 14: Actual Landing page 55](#_Toc219309355)

[Figure 15: Wireframe of Admin Login page 57](#_Toc219309356)

[Figure 16: Actual Admin login page 57](#_Toc219309357)

[Figure 17: Wireframe of Admin Dashboard 58](#_Toc219309358)

[Figure 18: Actual Admin Login Dashboard 58](#_Toc219309359)

[Figure 19: Wireframe of Family login portal 59](#_Toc219309360)

[Figure 20: Wireframe of Family Dashboardp 60](#_Toc219309361)

[Figure 21: Dialog Box to provide new prisoner detials 61](#_Toc219309362)

[Figure 22: New Prisoner being added 61](#_Toc219309363)

[Figure 23: Create new Prisoner Successfully 62](#_Toc219309364)

[Figure 24: Detail over view of all Prisoner 62](#_Toc219309365)

[Figure 25: Editable current details of prisoner when clicked edit button 63](#_Toc219309366)

[Figure 26: Status changed from active to released 63](#_Toc219309367)

[Figure 27: Updated details of prisoners 64](#_Toc219309368)

[Figure 28: DIalog Box to confirm deletion of prisoner details 65](#_Toc219309369)

[Figure 29: Deletion Confirmation Pop-up 65](#_Toc219309370)

[Figure 30: Prisoner with id 106 deleted 65](#_Toc219309371)

[Figure 31: Recently deleted prisoner stored in STACK 66](#_Toc219309372)

[Figure 32: Restore Confirmation 66](#_Toc219309373)

[Figure 33: Restored Deleted Prisoner and Stored back in QUEUE 66](#_Toc219309374)

[Figure 34: Add new prisoner with empty name 67](#_Toc219309375)

[Figure 35: Empty name error message displayed 68](#_Toc219309376)

[Figure 36: Error message for negative age 69](#_Toc219309377)

[Figure 37: Error message for negative sentence period 70](#_Toc219309378)

[Figure 38: Add Prisoner with invalide Date format 71](#_Toc219309379)

[Figure 39: Invalid Date Format Error message 72](#_Toc219309380)

**Table of Tables**

[Table 1: Method Description for PrisonerModel Class 12](#_Toc219303030)

[Table 2: Method Description for VisitRequestModel class 15](#_Toc219303031)

[Table 3: Method Description for AcitivityModel class 18](#_Toc219303032)

[Table 4: Method Description for MainFrame 21](#_Toc219303033)

[Table 5: Method Description for PrisonDialogHelper class 24](#_Toc219303034)

[Table 6: Method Descripton for TableButtonEditor 26](#_Toc219303035)

[Table 7: Method Descripton for TableButtonRenderer class 28](#_Toc219303036)

[Table 8: Method Description for TrashBinDialog Class 29](#_Toc219303037)

[Table 9: Method description for ViewDetailsDialog class 31](#_Toc219303038)

[Table 10: Method Description for VisitRequestsDialog class 33](#_Toc219303039)

[Table 11: Method description for CRUD class 38](#_Toc219303040)

[Table 12: Table description for OperationResult class 41](#_Toc219303041)

[Table 13: Method Description for PrisonController class 44](#_Toc219303042)

[Table 14: Method Description for SearchOperation 49](#_Toc219303043)

[Table 15: Method Description for Simple Queue 50](#_Toc219303044)

[Table 16: Method description for Simple Stack 52](#_Toc219303045)

[Table 17: Method Description for SortOperation 54](#_Toc219303046)

[Table 18: Method Description for TrashBinOperation 56](#_Toc219303047)

# 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 supporting competencies critical for both academic assessment and future professional development.

## Pupose

The main idea of the project involving the Prison Management System is the transformation of the theoretical ideas of data structure and algorithms into the real and socially applicable project. The project does not apply DSA as pure problem-solving means, but it strategically puts them into the context of operational realities of prison management where data is constantly updated by admissions, releases, transfers and changes in case status. Using linked lists to store records flexibly, stacks to support undo operations, queues to monitor activities and effective searching and sorting algorithms, the system demonstrates that the choices made during algorithm implementation have a direct influence on the performance, scalability, and reliability of the system. This is not only an academic intention; it contravenes the belief that small-scale educational systems are not relevant to the real worldFinally, it is meant to develop rigorous software design thinking where the data structure and algorithmic complexity and architectural designs like the MVC are deliberately chosen to meet the changing, real-world challenges.

## Audience

This project is intended to affect the undergraduate computing students, academic evaluators and novice software developers who need to know how their basic computer science concepts are relevant to non-textbook applications. It is especially applicable to students taking courses in data structures, algorithms, and specialty programming courses and who are prone to the difficulty of balancing theoretical complexity analysis with system behavior. Also, the project is conceptually important to policymakers, system designers and scholars who are concerned with the digitalization of correctional administration under resource constrained conditions. The system offers a way to engage technical audiences with the problem of management of prisons, under the guise that this is a problem of data, not necessarily one of administration, and in this manner the design decision of the software is likely to have a minimally harmful or even supportive impact on the actual institution-wide issues. This stance takes no shortcuts and, instead, encourages the reader to consider the effects of inefficient processes in critical areas.

## 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 become 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.

.

# 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**

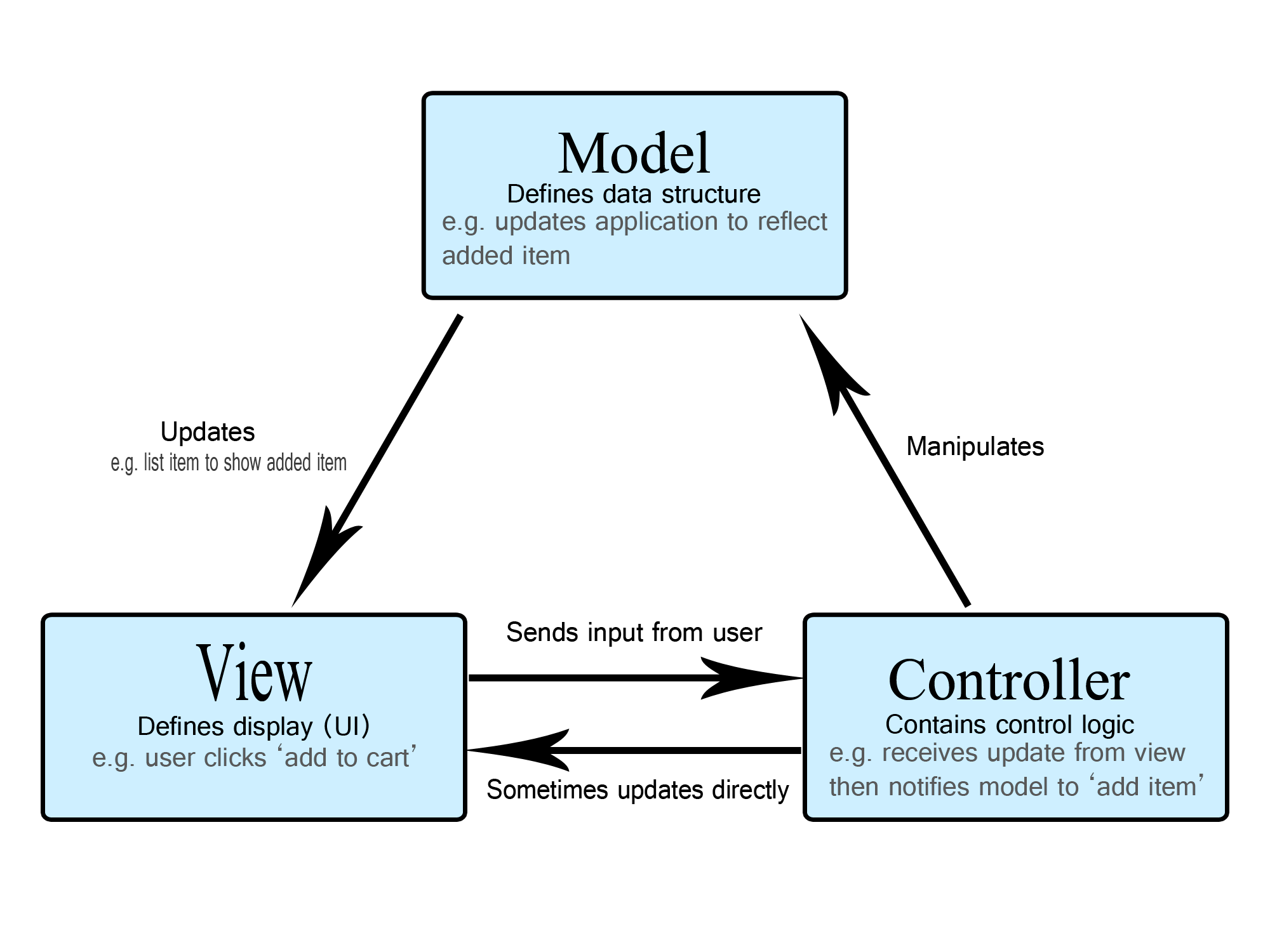
The Controller, exemplified by our PrisonController class, acts as an intermediary between the Model and View. It processes user requests, manipulates data through business logic operations (delegated to specialized classes like CRUD, SearchOperation, SortOperation, and TrashBinOperation), and updates 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 : MVC Architechture

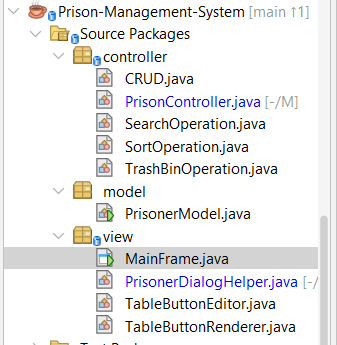


Figure : MVC Folder structure for Prison Management System

**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.).

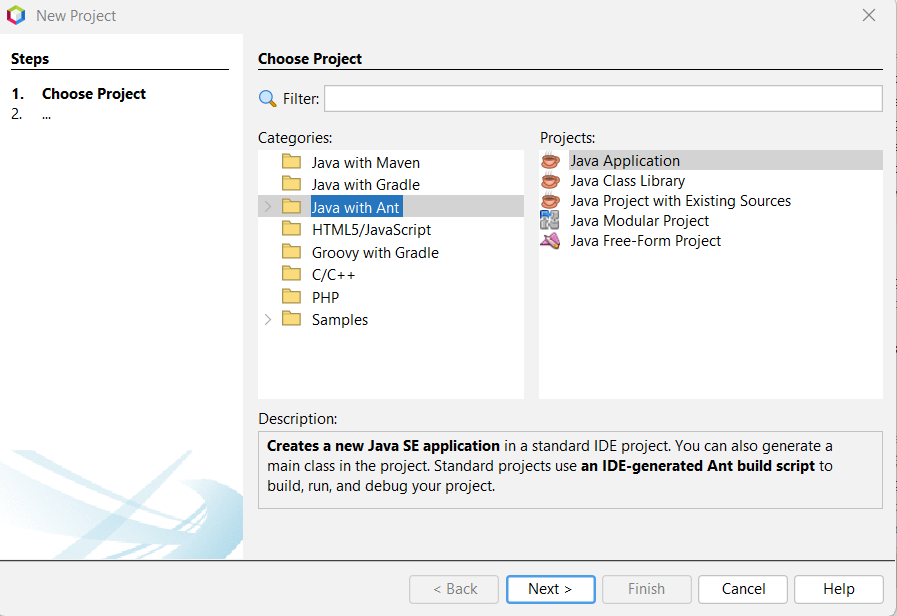


Figure : Ant Setup (1)

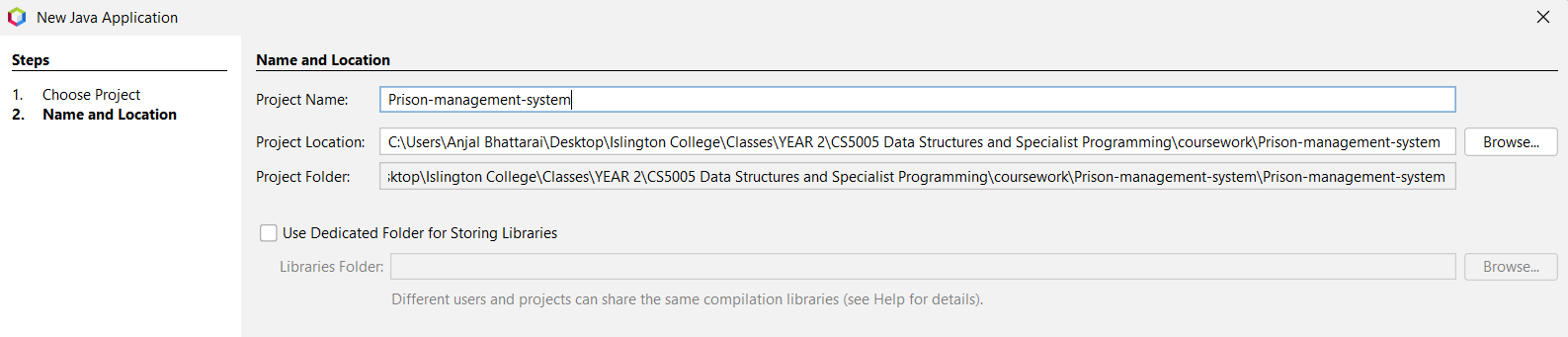


Figure : Ant Setup (2)

# Class Descriptions

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).

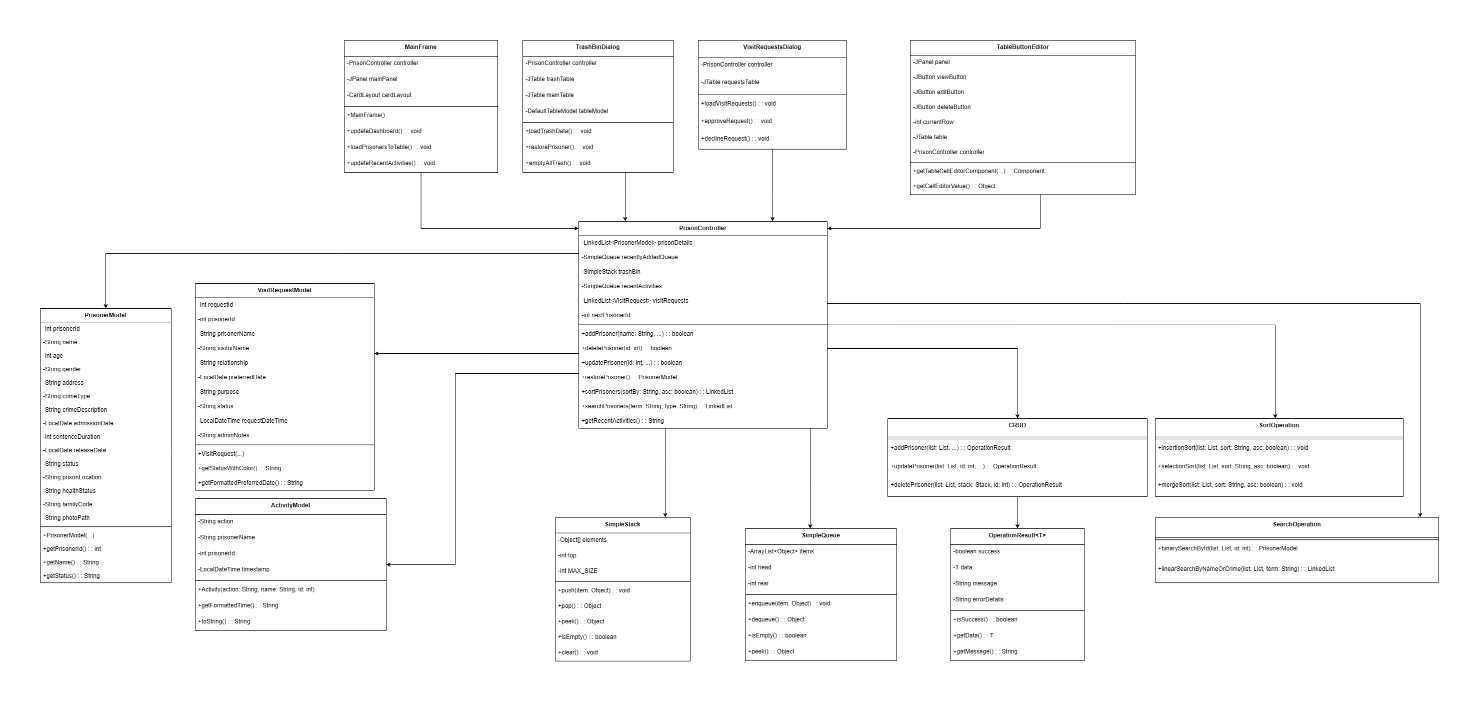


Figure : Complete Class diagram

This project implements a clear MVC-style Java Swing application where the model package defines domain objects (PrisonerModel, VisitRequest, Activity), the controller package encapsulates business logic and custom data structures (PrisonController, CRUD, SearchOperation, SortOperation, TrashBinOperation, SimpleQueue, SimpleStack) and the view package provides Swing-based UI and table rendering; CRUD.java exemplifies the separation of concerns by providing UI-agnostic, validated Create/Read/Update/Delete operations that return type-safe OperationResult objects, manage recent-addition and trash data structures, enforce input rules and ID generation, and log actions so the controller can perform activity tracking, visit-request handling and state changes without direct UI dependencies.

*[ \*Code Snippet of important methods < i.e CRUD, Searching and Sorting > are included in* [*Coursework Development*](#_CRUD_Implementation) *]*

Let’s dive into classes of each package:

## Model Package

**Overview**

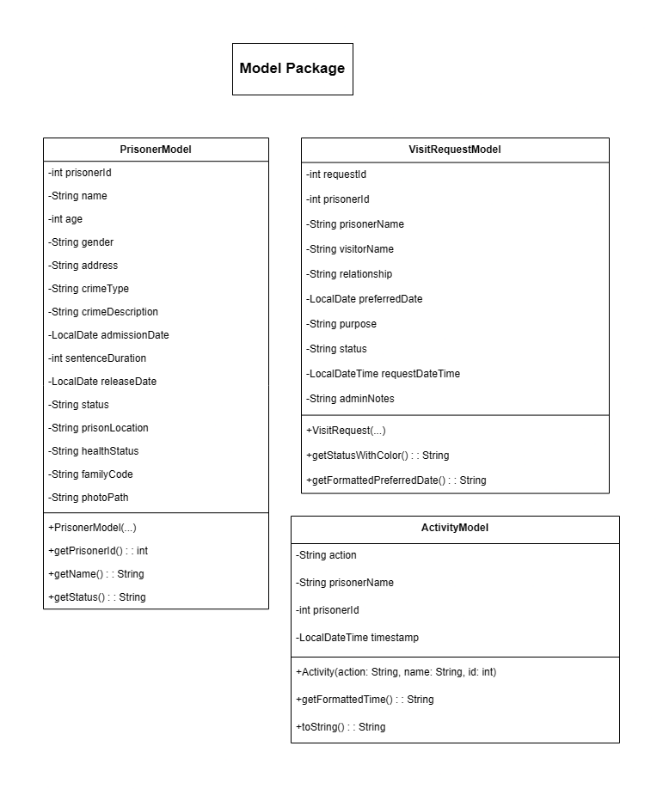
The model package contains POJO domain classes (PrisonerModel, VisitRequest, Activity) that encapsulate prisoner, visit-request, and activity data identifiers, timestamps, status fields and helpers such as release-date calculations used for validation, display and business logic. These classes expose simple getters/setters and small utility methods and are consumed by the controller layer for CRUD, search/sort, activity logging and UI rendering.

Figure : Model Package class diagram

**Classes on Model Package**

### 3.1.1 Prisoner Model

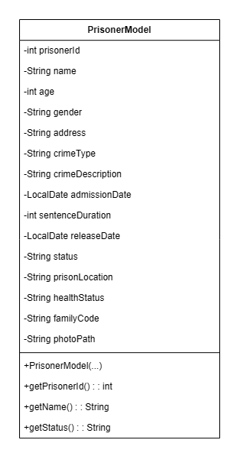
PrisonerModel is used by CRUD, PrisonController, SearchOperation, SortOperation, and TrashBinOperation.

Figure : Class Diagram for PrisonerModel

#### 3.1.1.1 Purpose

Represents a prisoner record and encapsulates prisoner state used across controller and view layers.

#### 3.1.1.2 Key attributes

* int prisonerId — unique identifier used for lookup and relations.
* String name, gender, address — primary identity and contact info.
* int age, int sentenceDuration — used for validation and release calculations.
* LocalDate admissionDate — start date for sentence / timeline logic.
* String crimeType, crimeDescription — domain details for filtering/search.
* String prisonLocation, familyCode, photoPath, status — UI/display and access control fields.
* String healthStatus — quick health summary shown in UI.

#### 3.1.1.3 Methods Description

Table : Method Description for PrisonerModel Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| PrisonerModel(...) | Constructor | Prisoner identity, crime details, admission date, sentence duration, prison location, family code, photo path | Creates a new prisoner entity by initializing personal, legal, and administrative attributes. The release date is automatically calculated using the admission date and sentence duration, ensuring data consistency at the time of object creation. |
| PrisonerModel(...) | Constructor | Same as above plus status | Initializes a prisoner entity while allowing the status to be explicitly defined. This supports scenarios such as loading historical records or externally validated prisoner data. |
| setAdmissionDate(LocalDate) | Business Logic Method | Admission date | Updates the prisoner’s admission date and automatically recalculates the release date to maintain logical consistency between sentence duration and incarceration period. |
| setSentence Duration(int) | Business Logic Method | Sentence duration (months) | Updates the length of the sentence and recalculates the expected release date, preventing inconsistencies in incarceration timelines. |
| main(String[] args) | Test Method | Command-line arguments | Serves as a basic validation mechanism to demonstrate object creation and verify correct calculation of derived attributes such as release date and status. |

### 3.1.2 Visit Request Model

VisitRequestModel is used by PrisonController, Visit UI.



Figure : Class Diagram for VisitRequestModel

#### 3.1.2.1 Purpose

Models a family/visitor visit request tied to a prisoner; used by controllers and UI to list, approve/deny and track requests.

#### 3.1.2.2 Key attributes

* int requestId — unique for each request.
* int prisonerId, String prisonerName — link to prisoner and display.
* String visitorName, relationship, purpose — visitor details and reason.
* LocalDate preferredDate — requested visit date.
* String status — e.g., "Pending", "Approved", "Denied".
* String adminNotes — admin feedback or scheduling notes.
* LocalDateTime createdAt — timestamp for ordering/audit.

#### 3.1.2.3 Methods Description

Table : Method Description for VisitRequestModel class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| VisitRequest  (...) | Constructor | PrisonerID,  prisonername, visitor name, relationship, preferred date, purpose | Creates a new family visit request with an auto-generated request ID. The request status is initialized as *Pending*, the request timestamp is recorded automatically, and administrative notes are initialized as empty. |
| VisitRequest  (...) | Constructor | All  request attributes | Initializes a visit request using pre-existing data, typically when loading records from persistent storage. The internal request ID counter is updated to prevent identifier duplication. |
| getFormatted  RequestDate  Time() | Presentation Logic Method | — | Returns the request submission date and time in a standardized human-readable format suitable for display in user interfaces. |
| getFormatted  Preferred  Date() | Presentation  Logic Method | — | Formats the preferred visit date into a consistent date string for display and reporting purposes. |
| getStatus  With  Color() | UI Support Method | — | Returns a color-coded representation of the request status to visually distinguish *Pending*, *Approved*, and *Declined* states in graphical interfaces. |
| toString() | Override Method | — | Provides a concise textual summary of the visit request, combining prisoner details, visit date, and request status for logging and debugging purposes. |

### 3.1.3 Activity Model

ActivityModel is used by PrisonController (recentActivities).

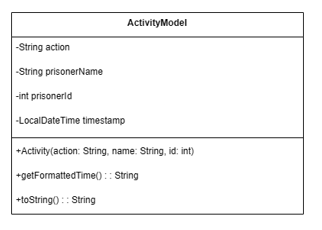


Figure : Class Diagram for ActivityModel

#### 3.1.3.1 Purpose

Lightweight audit/event record for recent actions (ADD, UPDATE, DELETE, RESTORE, VIEW) shown in UI activity panel.

#### 3.1.3.2 Key attributes

* String action — action type label used for color/formatting.
* String prisonerName — contextual subject.
* int prisonerId — link back to prisoner.
* LocalDateTime timestamp — when action occurred.

#### 3.1.3.3 Methods Description

Table : Method Description for AcitivityModel class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| Activity(String action, String prisonerName, int prisonerId) | Constructor | Action type, prisoner name, prisoner ID | Creates a new activity record representing a system event. The timestamp is automatically generated at creation time to ensure accurate and tamper-resistant activity logging. |
| getFormattedTime() | Presentation Logic Method | — | Formats the activity timestamp into a concise time string suitable for display in dashboards or recent-activity logs. |
| toString() | Override Method | — | Generates a human-readable summary of the activity, combining time, action type, prisoner identity, and identifier for logging and user interface display. |

## 3.2 View Package

**Overview**The view package contains UI components and helpers that render prisoner data and handle user interactions. It supplies the main application window (MainFrame), dialog helpers for creating/editing records and viewing details, table cell renderers/editors for action buttons, and specialized dialogs for the trash bin and visit requests. These classes focus on layout, user-event handling, table population, and delegating actions to the controller layer.

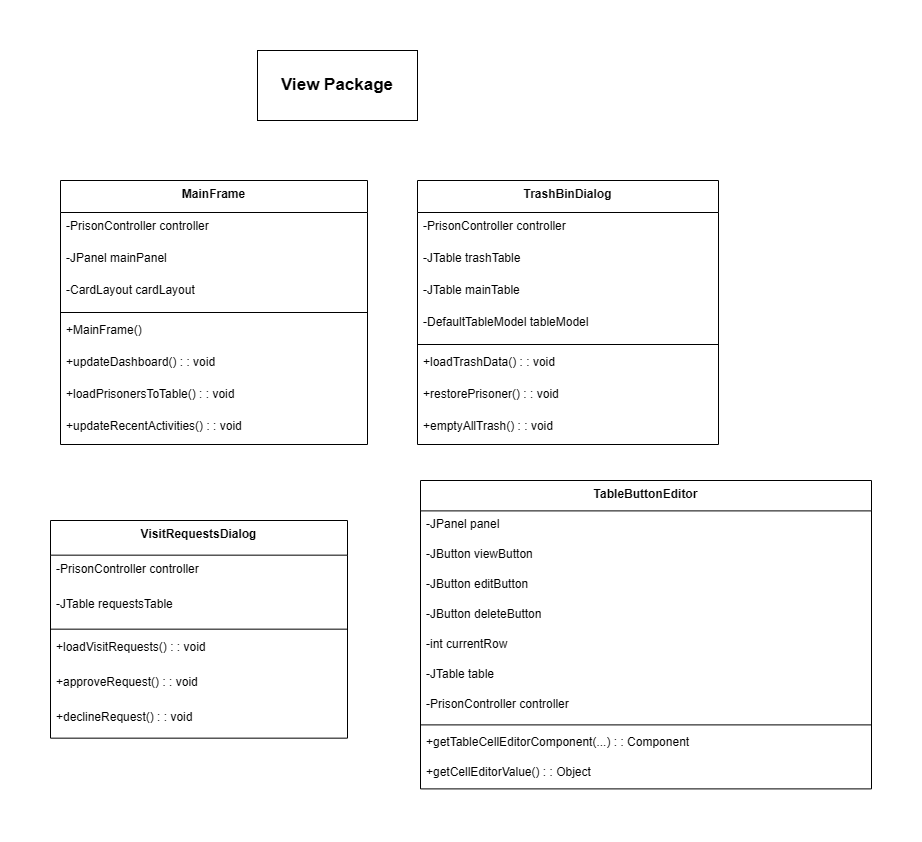


Figure : View Package Class Diagram

**Classes in View Package**

There are 7 classes in view packages. Each class are well described below.

### 3.2.1 MainFrame

This is file we run for the first time.

#### 3.2.1.1 Purpose

Hosts the primary application window, panels, search/sort controls, prisoner table and orchestrates opening dialogs.

#### 3.2.1.2 Key attributes

* private static final Logger logger — class logger.
* Panel name constants: HOME\_PANEL, ADMIN\_LOGIN\_PANEL, ADMIN\_DASHBOARD\_PANEL, FAMILY\_LOGIN\_PANEL, FAMILY\_DASHBOARD\_PANEL.
* Color scheme constants: PRIMARY\_COLOR, SECONDARY\_COLOR, ACCENT\_COLOR, DANGER\_COLOR, WARNING\_COLOR, BACKGROUND\_COLOR, CARD\_COLOR, TEXT\_PRIMARY, TEXT\_SECONDARY.
* private PrisonController controller — shared controller instance.
* private int currentFamilyPrisonerId — ID used by family dashboard.
* Swing components (selected): JPanel mainPanel, JTable PrisonerRecordTable, search/sort controls (SearchTextField, SearchTypeComboBox, SearchButton, SortByComboBox, SortButton, etc.), recent-activity label ActivityLabel, and various panels/fields used across cards.

#### 3.2.1.3 Methods Description

Table : Method Description for MainFrame

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| MainFrame() | Constructor | — | Creates and initializes the main application window. Sets up the custom components, initializes the controller with data, and displays the home panel. |
| loadFamily  Dashboard  (Prisoner  Model  prisoner) | Private Helper Method | prisoner (PrisonerModel) | Populates the Family Dashboard panel with the details of a specific prisoner. Updates labels, sets up relationship dropdown, initializes placeholder text for form fields, loads existing visit requests, and forces UI refresh. |
| loadVisit  Requests  ForPrisoner  (int prisonerId) | Private Helper Method | prisonerId (int) | Queries the controller for all visit requests associated with the given prisoner ID and loads them into the dashboard's table for display. Shows a placeholder message if no requests exist. |
| clearFamily  Dashboard() | Private Helper Method | — | Resets all prisoner information labels and form fields on the Family Dashboard to default/empty values and clears the visit request table. |
| showFamily  Dashboard  Panel() | Panel Navigation Method | — | Uses a CardLayout to switch the main panel's view to the Family Dashboard panel. |
| Setup  Placeholder  Text(JTextFieldtextField,  String placeholder) | Private UI Utility Method | textField (JTextField), placeholder (String) | Configures a given text field to display placeholder text (in gray) when empty. Adds focus listeners to clear the placeholder on focus and restore it if the field is left empty. |
| setupCustom  Components() | Private Initialization Method | — | Configures event listeners and behaviors for various UI components after they are initialized. This includes setting up search, refresh, sort, and visit request buttons, as well as navigation buttons for the login panels. |
| Update  Recent  Activities() | Public Data Update Method | — | Fetches the latest formatted activity log from the controller and updates the Recent Activity panel's label to display it. |
| showHome  Panel() | Panel  Navigation Method | — | Switches the main panel's view to the Home Panel using CardLayout. Also updates the home statistics before showing the panel. |
| Update  Home  Statistics() | Private Data Update Method | — | Calculates key prison statistics (total prisoners, gender counts, occupancy rate) from the controller's data and updates the corresponding labels on the Home Panel with formatted HTML. |
| Show  Admin  LoginPanel() | Panel  Navigation Method | — | Uses CardLayout to switch the main panel's view to the Admin Login panel. |
| Show  Admin  Dashboard  Panel() | Pane  l Navigation Method | — | Uses CardLayout to switch the main panel's view to the Admin Dashboard panel. |
| ShowFamilyLoginPanel() | Panel  Navigation Method | — | Uses CardLayout to switch the main panel's view to the Family Login panel. |
| ShowFamily  Dashboard  Panel(int prisonerId) | Panel  Navigation Method | prisonerId (int) | Sets the current family prisoner ID and then uses CardLayout to switch the main panel's view to the Family Dashboard panel. |
| Main(String args[]) | Static Main Method | args (String[]) | The application's entry point. Sets the Nimbus look-and-feel and launches the MainFrame on the Event Dispatch Thread (EDT). |

### 3.2.2 Prisoner Dialog Helper

#### 3.2.2.1 Purpose

Static helper that builds Add/Edit prisoner dialogs, performs input validation, previews photos, and invokes controller methods.

#### 3.2.2.2 Key Attributes

* None

**3.2.2.3 Method Description**

Table : Method Description for PrisonDialogHelper class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| showAddDialog(PrisonController controller, JFrame parent, JTable table) | Static Dialog Method | Controller instance, parent frame, prisoner table | Opens a form for adding a new prisoner. Handles field creation, input validation (age, admission date, sentence duration, mandatory fields), photo selection, and updates the prisoner table and recent activities upon successful submission. |
| showEditDialog(PrisonerModel prisoner, PrisonController controller, JFrame parent, JTable table) | Static Dialog Method | Prisoner object, controller, parent frame, prisoner table | Opens a pre-filled form to edit an existing prisoner's details. Validates inputs, handles photo updates, updates the prisoner table, and refreshes recent activities upon successful submission. |
| createFormPanel(JTextField nameField, JSpinner ageSpinner, JComboBox<String> genderCombo, JTextField addressField, JTextField crimeTypeField, JScrollPane crimeDescScroll, JSpinner admissionDateSpinner, JSpinner sentenceSpinner, JTextField locationField, JComboBox<String> statusCombo, JTextField familyCodeField, JLabel photoPreviewLabel, JButton choosePhotoBtn) | Private Panel Builder | Swing components for form fields | Constructs a JPanel using GridBagLayout that organizes all prisoner form fields and photo components in a consistent layout for Add/Edit dialogs. |
| setupTableButtons(JTable table, PrisonController controller, JFrame parent) | Static Table Configuration | JTable, controller, parent frame | Configures prisoner table with row height, column widths, and sets up custom button renderers and editors for actions (View, Edit, Delete) to ensure a user-friendly interface. |

### 3.2.3 Table Button Editor

#### 3.2.3.1 Purpose

Cell editor that renders clickable View/Edit/Delete buttons inside a table cell and routes clicks to handlers that call controller/view dialogs.

#### 3.2.3.2 Key Attributes

* JPanel panel, JButton viewButton, JButton editButton, JButton deleteButton — UI components.
* int currentRow — currently edited row.
* JTable table, PrisonController controller, JFrame parentFrame — context for actions.

#### 3.2.3.3 Methods Description

Table : Method Descripton for TableButtonEditor

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Component** | **Type** | **Key Parameters** | **Description** |
| TableButtonEditor(JCheckBox checkBox, JTable table, PrisonController controller, JFrame parentFrame) | Constructor | CheckBox (for DefaultCellEditor), prisoner table, controller, parent frame | Initializes the custom table cell editor with three action buttons: View, Edit, and Delete. Configures button appearance, event listeners, and layout inside a panel. |
| getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column) | Overridden method | Table, cell value, row index, column index | Returns the button panel as the editor component for the table cell. Tracks the current row for button actions. |
| getCellEditorValue() | Overridden method | None | Returns the current cell value when editing stops (empty string in this implementation). |
| stopCellEditing() | Overridden method | None | Stops cell editing and commits any changes. |
| setupTableButtons() | Private helper method | None (uses class fields) | Resets the table’s action column renderer and editor after data changes to maintain interactive buttons in the table. |
| viewButton ActionListener | Button event handler | None | Fetches the prisoner ID from the selected row and opens a ViewDetailsDialog for the prisoner. Displays error dialogs if ID is missing or prisoner not found. |
| editButton ActionListener | Button event handler | None | Fetches the prisoner ID from the selected row and invokes PrisonerDialogHelper.showEditDialog() to edit prisoner details. Includes debug logging and error handling for missing or invalid IDs. |
| deleteButton ActionListener | Button event handler | None | Fetches the prisoner ID and deletes the prisoner via the controller. Refreshes the table and updates recent activities. Handles errors gracefully. |

### 3.2.4 Table Button Renderer

#### 3.2.4.1 Purpose

Cell renderer that displays an action button (or icon) in a table cell without being editable; provides consistent visual styling.

#### 3.2.4.2 Attributes

* JButton viewButton, JButton editButton, JButton deleteButton.

#### 3.2.4.2 Methods Description

Table : Method Descripton for TableButtonRenderer class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Component** | **Type** | **Key Parameters** | **Description** |
| TableButtonRenderer() | Constructor | None | Initializes the custom table cell renderer panel with three buttons: View, Edit, and Delete. Sets their preferred size, background colors, foreground colors, focus behavior, and border properties. Adds buttons to the panel using FlowLayout. |
| getTableCellRenderer  Component  (JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) | Overridden method | Table, cell value, selection status, focus status, row index, column index | Returns the panel as the renderer for the table cell. Dynamically adjusts the panel’s background based on row selection. Ensures consistent visual representation of the action buttons in each row of the table. |

### 3.2.5 Trash Bin Dialog

#### 3.2.5.1 Purpose

Dialog listing soft-deleted prisoners from the SimpleStack trash; supports restore (pop) and empty (clear) actions.

#### 3.2.5.2 Key Attributes

* PrisonController controller, JTable trashTable, DefaultTableModel tableModel, JLabel statusLabel, JFrame parentFrame, JTable mainTable.

#### 3.2.5.3. Methods Description

Table : Method Description for TrashBinDialog Class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Component** | **Type** | **Key Parameters** | **Description** |
| TrashBinDialog(JFrame parent, PrisonController controller, JTable mainTable) | Constructor | Parent frame, controller, main table | Initializes the dialog for viewing the trash bin. Sets up layout, initializes components, loads trash data, and centers the dialog relative to the parent. |
| initComponents() | Private method | None | Builds the UI of the dialog: header panel, trash table, and button panel. Configures fonts, colors, table column widths, and button actions (Restore, Empty, Refresh, Close). Ensures proper visual hierarchy and user-friendly layout. |
| loadTrashData() | Private method | None | Loads deleted prisoners from the SimpleStack trash bin into the table. Displays prisoners from top (most recently deleted) to bottom (oldest), updates statusLabel, and handles empty trash gracefully. |
| restoreTopPrisoner() | Private method | None | Restores the most recently deleted prisoner (stack pop operation) from trash back to the main table. Refreshes both tables, resets table buttons, and updates recent activity logs. Handles empty trash scenario with user feedback. |
| emptyAllTrash() | Private method | None | Clears all prisoners from the trash (stack clear operation). Refreshes table, updates activity logs, and shows informational messages if trash is already empty. |

### 3.2.6 View Details Dialog

#### 3.2.6.1 Purpose

Read-only dialog that shows full prisoner details, photo preview, and provides an Edit action.

#### 3.2.6.2 Key Attributes

* PrisonerModel prisoner, PrisonController controller, JTable table, JFrame parentFrame.
* UI elements created inside initComponents (photo label, details panel, status label, action buttons).

#### 3.2.6.3 Methods Description

Table : Method description for ViewDetailsDialog class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Component** | **Type** | **Key Parameters** | **Description** |
| ViewDetailsDialog(JFrame parent, PrisonerModel prisoner, PrisonController controller, JTable table) | Constructor | Parent frame, prisoner object, controller, main table | Initializes the dialog to display complete prisoner details. Sets up the UI layout, loads the prisoner’s photo, and centers the dialog relative to the parent. |
| initComponents() | Private method | None | Builds the full UI: photo panel, details panel, and button panel. Configures fonts, colors, scrollable panels, and action buttons (Edit, Close). Handles photo loading with debug checks, scales images, and shows fallback messages if photo is missing or invalid. |
| addField(JPanel panel, GridBagConstraints gbc, int row, String label, String value) | Private helper method | Panel, layout constraints, row index, field label, field value | Adds a labeled data field to the details panel using GridBagLayout. Ensures consistent font and alignment for all prisoner attributes. |
| Photo Panel Handling | UI Component | None | Displays the prisoner’s photo with border and scaling. Handles missing photos or invalid paths by showing appropriate messages and debug logs. |
| Details Panel | UI Component | None | Shows all prisoner details: ID, Name, Age, Gender, Address, Crime Type & Description, Admission/Release Dates, Sentence, Location, Status (with color coding), Health Status, and Family Code. Wraps long fields (like Crime Description) in scrollable text areas. |
| Status Label Color Coding | UI Feature | Prisoner status string | Dynamically sets background/foreground color based on status: Active (green), Released (blue), Medical (yellow), Solitary (red), Default (gray). Improves visual recognition of prisoner status. |
| Button Panel | UI Component | None | Contains **Edit Prisoner** (opens edit dialog via PrisonerDialogHelper) and **Close** buttons. Ensures consistent styling and user-friendly layout. |
| Scrollable Details | UI Feature | None | Wraps the details panel in a JScrollPane to accommodate long text content and prevent layout overflow. Ensures full visibility for all prisoner information. |

### 3.2.7 Visit Requests Dialog

#### 3.2.7.1 Purpose

Admin UI for managing family visit requests; lists requests and provides Approve/Decline actions with notes.

#### 3.2.7.2 Key Attributes

* PrisonController controller, JTable requestsTable, DefaultTableModel tableModel, JLabel statusLabel.
* Color constants for consistent styling: PRIMARY\_COLOR, ACCENT\_COLOR, DANGER\_COLOR, BACKGROUND\_COLOR, CARD\_COLOR.

#### 3.2.7.3 Methods Description

Table : Method Description for VisitRequestsDialog class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Component** | **Type** | **Key Parameters** | **Description** |
| VisitRequestsDialog(Frame parent, PrisonController controller) | Constructor | Parent frame, PrisonController instance | Initializes the dialog for managing family visit requests. Sets up UI components, loads all requests into the table, and centers dialog relative to parent. |
| initComponents() | Private method | None | Builds the complete UI: header panel, status panel, main table with actions column, and bottom buttons. Configures fonts, colors, borders, table column widths, scroll panes, and adds hover effects to buttons. |
| setupActionButtons() | Private method | None | Attaches custom ButtonRenderer and ButtonEditor to the "Actions" column of the requests table, enabling approve/decline buttons for each row. |
| loadVisitRequests() | Private method | None | Fetches all visit requests from the controller. Populates the table row-by-row with request data (Request ID, Prisoner Info, Visitor Info, Dates, Purpose, Status). Updates statusLabel to show total, pending, and processed requests. |
| refreshTable() | Public method | None | Refreshes the table by calling loadVisitRequests(). Used after an approve or decline action to update the UI. |
| createStyledButton(Strng text,  Color bgColor) | Private helper method | Button text, background color | Creates a consistent, styled JButton with hover effect, font, size, cursor pointer, and color scheme. Used for bottom panel buttons (Refresh, Close). |
| **ButtonRenderer** | Inner class | None | Extends JPanel and implements TableCellRenderer. Displays **Approve** and **Decline** buttons in the "Actions" column. Buttons are enabled only if the request status is "Pending". Ensures consistent layout and styling for all rows. |
| **ButtonEditor** | Inner class | JCheckBox, PrisonController, VisitRequestsDialog | Extends DefaultCellEditor. Handles click events for **Approve** and **Decline** buttons per row. Calls controller to update request status and displays confirmation dialogs. Refreshes table after actions. |
| handleApprove() | Private method inside ButtonEditor | None | Prompts admin for optional notes. Marks the selected visit request as "Approved" via the controller. Shows confirmation message and refreshes table. |
| handleDecline() | Private method inside ButtonEditor | None | Prompts admin to enter a reason for declining. Marks the selected visit request as "Declined" via the controller. Shows confirmation message and refreshes table. Requires non-empty reason to proceed. |
| Status Column Renderer | UI Feature | None | Custom DefaultTableCellRenderer for the Status column. Colors text based on request status: Approved (green), Declined (red), Pending (orange). Centers text and applies bold font. |
| Table Column Configuration | UI Feature | None | Sets preferred widths for all columns and row height for better readability. "Actions" column is editable to hold buttons; other columns are read-only. |
| Header Panel | UI Component | None | Displays dialog title with primary color background. Matches main application theme for consistency. |
| Status Panel | UI Component | None | Shows dynamic summary: total requests, pending requests, processed requests. Styled with borders and padding for clarity. |
| Bottom Panel Buttons | UI Component | None | Provides **Refresh** and **Close** buttons with hover effects and consistent styling. Refresh reloads the table; Close disposes the dialog. |

## 3.3 Controller Package

**Overview**

The controller package contains classes that implement application logic: CRUD operations, high-level orchestration via PrisonController, search/sort/trash behaviors, simple data structures used by operations, and small result wrappers. Controllers mediate between the view layer and model layer, enforce validation rules, persist or mutate data stores (in-memory or file-backed), and produce OperationResult objects for UI feedback.

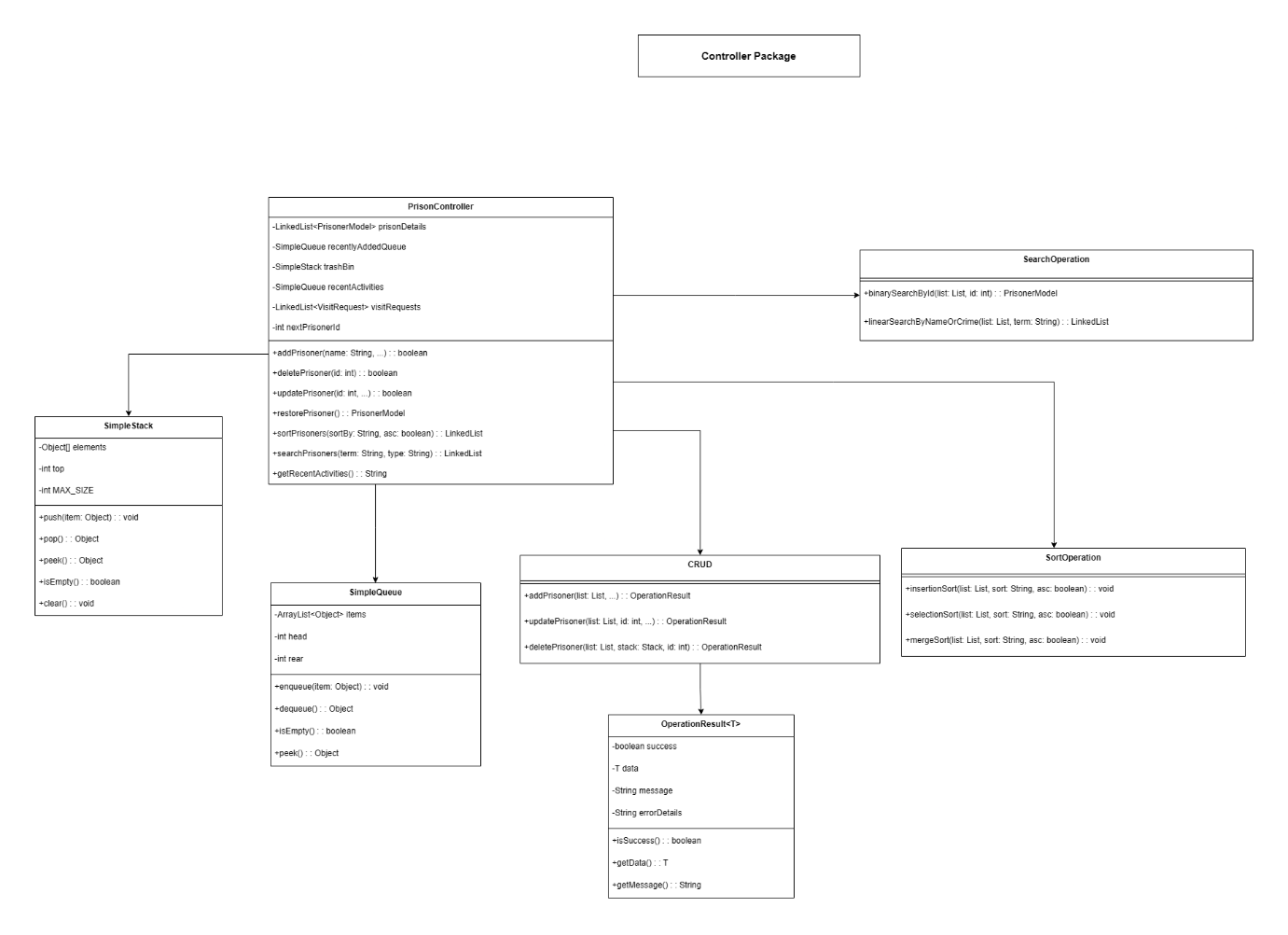


Figure : Controller Package Class Diagrams

**Classes on Controller Package**

### 3.3.1 CRUD

#### 3.3.1.1 Purpose

Encapsulates create, read, update and soft-delete operations against the prisoner data store; used by higher-level controllers and UI actions.

#### 3.3.1.2 Key attributes

* private static final int MAX\_RECENT : maximum number of recently-added items tracked.

#### 3.3.1.3 Method Description

Table : Method description for CRUD class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| addPrisoner (LinkedList <PrisonerModel>  prisonDetails,  SimpleQueue recentlyAddedQueue,  int nextPrisonerId,  String name, int age,  String gender,  String address,  String crimeType, String crimeDescription,  LocalDate admissionDate,  int sentenceDuration,  String prisonLocation,  String familyCode, String photoPath, String status) | Static Create Method | prisonDetails (LinkedList<PrisonerModel>), recentlyAddedQueue (SimpleQueue), nextPrisonerId (int), name (String), age (int), gender (String), address (String), crimeType (String), crimeDescription (String), admissionDate (LocalDate), sentenceDuration (int), prisonLocation (String), familyCode (String), photoPath (String), status (String) | Adds a new prisoner to the system with validated input. Generates a unique prisoner ID, adds the prisoner to the main list, updates the recent additions queue, and returns an OperationResult indicating success or failure. |
| getPrisonerById (LinkedList <PrisonerModel>  prisonDetails, int id) | Static Read Method | prisonDetails (LinkedList<PrisonerModel>), id (int) | Retrieves a prisoner object from the main list by their unique ID. Returns null if no prisoner is found. |
| updatePrisoner (LinkedList <PrisonerModel>  prisonDetails,  int prisonerId,  String name, int age,  String gender,  String address,  String crimeType,  String crimeDescription,  LocalDate admissionDate,  int sentenceDuration, String prisonLocation,  String familyCode, String photoPath) | Static Update Method | prisonDetails (LinkedList <PrisonerModel>), prisonerId (int), name (String), age (int), gender (String), address (String), crimeType (String), crimeDescription (String), admissionDate (LocalDate), sentenceDuration (int), prisonLocation (String), familyCode (String), photoPath (String) | Updates the information of an existing prisoner after validating inputs and ensuring no duplicate names. Returns an OperationResult indicating whether the update was successful. |
| deletePrisoner (LinkedList <PrisonerModel>  prisonDetails,  SimpleStack trashBin, int prisonerId) | Static Delete Method | prisonDetails (LinkedList <PrisonerModel>), trashBin (SimpleStack), prisonerId (int) | Removes a prisoner from the main list and pushes them to a trash stack for potential restoration. Returns an OperationResult with the deleted prisoner’s data or an error message if deletion fails. |
| getNextAvailableId (LinkedList <PrisonerModel> prisonDetails, int nextPrisonerId) | Static Helper Method | prisonDetails (LinkedList <PrisonerModel>), nextPrisonerId (int) | Calculates the next available prisoner ID by comparing the maximum current ID with a tracker value. Ensures unique ID generation for new prisoners. |
| getRecentActivities (SimpleQueue recentlyAdded Queue) | Static Helper Method | recentlyAddedQueue (SimpleQueue) | Generates a formatted HTML string listing the most recent prisoner additions or sample activities if none exist. Used to update activity display panels in the UI. |

### 3.3.2 OperationResult

#### 3.3.2.1 Purpose

Generic wrapper for operation outcomes (success/failure) with optional payload and error details.

#### 3.3.2.1 Key attributes

* private final boolean success : success flag.
* private final T data : optional payload on success.
* private final String message : human-readable message.
* private final String errorDetails : optional detailed error information.Methods

#### 3.3.2.3 Method Description

Table : Table description for OperationResult class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| OperationResult (boolean success, T data, String message, String errorDetails) | Private Construc-tor | success (boolean), data (T), message (String), errorDetails (String) | Initializes an immutable result object representing either a successful or failed operation. Access is restricted to enforce usage through static factory methods. |
| success(T data, String message) | Static Factory Method | data (T), message (String) | Creates a successful OperationResult containing returned data and a descriptive success message. Used when an operation completes successfully and produces a result. |
| success(String message) | Static Factory Method | message (String) | Creates a successful OperationResult without associated data. Suitable for operations that succeed but do not return a value. |
| failure(String message) | Static Factory Method | message (String) | Creates a failed OperationResult with a user-friendly error message and no detailed error information. |
| failure(String message, String errorDetails) | Static Factory Method | message (String), errorDetails (String) | Creates a failed OperationResult with both a general error message and detailed technical information for debugging or logging. |
| isSuccess() | Public Query Method | — | Returns true if the operation completed successfully; otherwise returns false. |
| isFailure() | Public Query Method | — | Returns true if the operation failed. Acts as a semantic inverse of isSuccess() for improved readability. |
| getData() | Public Accessor Method | — | Returns the data associated with a successful operation, or null if no data was returned or the operation failed. |
| getMessage() | Public Accessor Method | — | Returns the main success or error message associated with the operation result. |
| getErrorDetails() | Public Accessor Method | — | Returns detailed error information intended for debugging or logging purposes, or null if not provided. |
| getFullErrorMessage() | Public Utility Method | — | Returns a combined error message that includes both the main message and detailed error information when available. Useful for consistent error reporting in the UI or logs. |
| toString() | Over-ridden Utility Method | — | Returns a formatted string representation of the operation result, clearly indicating success or failure along with any message and associated data. |

### 3.3.3 PrisonController

#### 3.3.3.1 Purpose

High-level facade used by the UI to perform workflows (add/edit/delete/restore prisoners, manage visit requests, load data into tables, and log activities).Key attributes

#### 3.3.3.2 Key Attributes

* private LinkedList<PrisonerModel> prisonDetails: main inmemory prisoner list.
* private SimpleQueue recentlyAddedQueue : FIFO queue for recent additions (used by recent activities display).
* private SimpleStack trashBin — LIFO stack storing deleted prisoners for restoration.
* private SimpleQueue recentActivities — queue holding Activity entries.
* private LinkedList<VisitRequest> visitRequests — stored visit requests.
* private static final int MAX\_ACTIVITIES — maximum activities to retain.
* private int nextPrisonerId — next ID tracker (starts at 101).Methods

#### 3.3.3.3 Method Description

Table : Method Description for PrisonController class

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| PrisonController() | Constructor | — | Initializes the controller, prepares core data structures, and loads sample prisoner data with Nepali context for demonstration and testing purposes. |
| addPrisoner(...) | Public Controller Method | Prisoner details | Handles prisoner creation requests from the UI layer. Delegates validation and creation logic to CRUD.addPrisoner, updates the next available ID, and logs the activity if successful. |
| getNextAvailableId() | Public Utility Method | — | Returns the next available prisoner ID for UI preview by delegating to CRUD.getNextAvailableId. |
| getRecentlyAdded() | Public View Helper | — | Retrieves formatted recent prisoner activity for dashboard display by delegating to CRUD.getRecentActivities. |
| getPrisonerById(int id) | Public Query Method | id (int) | Retrieves a prisoner record matching the given ID by delegating to the CRUD layer. |
| getAllPrisoners() | Public Data Accessor | — | Returns the full list of prisoners currently managed by the system. |
| updatePrisoner(...) | Public Controller Method | Prisoner ID and updated fields | Updates an existing prisoner record via CRUD.updatePrisoner. Logs an update activity if the operation succeeds. |
| deletePrisoner(int prisonerId) | Public Controller Method | prisonerId (int) | Deletes a prisoner by delegating to CRUD.deletePrisoner, moves the record to a trash stack, logs the action, and displays error feedback if deletion fails. |
| restorePrisoner() | Public Recovery Method | — | Restores the most recently deleted prisoner from the trash stack and logs the restoration activity. |
| getTrashContents() | Public View Helper | — | Returns an HTML-formatted representation of deleted prisoners currently in the trash bin. |
| getTrashSize() | Public Query Method | — | Returns the number of prisoners currently stored in the trash bin. |
| emptyTrash() | Public Maintenance Method | — | Permanently removes all prisoners from the trash bin and logs the action if any records were deleted. |
| getTrashBin() | Public Accessor Method | — | Provides access to the trash stack for advanced viewing or debugging purposes. |
| logActivity(String action, String name, int id) | Private Helper Method | Activity details | Records system activities (add, update, delete, restore) into a bounded queue for dashboard display and auditing. |
| getRecentActivitiesQueue() | Public Accessor Method | — | Returns the queue containing recent system activities. |
| getFormattedActivities() | Public View Helper | — | Generates an HTML-formatted activity log with timestamps and color-coded actions for UI dashboards. |
| getColorForAction(String action) | Private Utility Method | action (String) | Maps activity types to UI color codes to improve visual clarity in activity logs. |
| searchPrisoners(String type, String term) | Public Search Method | Search type and query | Searches prisoner records based on selected criteria by delegating to SearchOperation. |
| sortPrisoners(String sortBy, boolean asc) | Public Sorting Method | Sort field and order | Sorts prisoner records using default sorting logic via SortOperation. |
| sortPrisoners(String sortBy, boolean asc, String algorithm) | Public Sorting Method | Field, order, algorithm | Sorts prisoners using a user-selected sorting algorithm such as Insertion Sort, Selection Sort, or Merge Sort. |
| loadPrisonerListToTable(...) | Public UI Loader | JTable, prisoner list | Loads a filtered or sorted list of prisoners into a Swing table model for display. |
| getPrisonerCount() | Public Query Method | — | Returns the total number of prisoners currently in the system. |
| prisonerExists(int id) | Public Validation Method | id (int) | Checks whether a prisoner with the specified ID exists in the system. |
| loadPrisonerToTable(JTable table) | Public UI Loader | JTable | Loads all prisoners into the table without filtering or sorting. |
| getAbsoluteImagePath(String fileName) | Private Utility Method | Image file name | Resolves and returns the absolute file path for prisoner image resources. |
| loadSampleNepalData() | Private Initialization Method | — | Populates the system with realistic, diverse prisoner records reflecting Nepali demographics, crimes, health statuses, and prison locations. |
| validateFamilyLogin(int id, String code) | Public Authentication Method | Prisoner ID, family code | Validates family portal login credentials and logs successful access attempts. |
| addVisitRequest(...) | Public Controller Method | Visit request details | Creates and stores a new visit request for a prisoner, including visitor information and purpose. |
| getAllVisitRequests() | Public Data Accessor | — | Returns all visit requests submitted to the system. |
| getVisitRequestsForPrisoner(int id) | Public Query Method | prisonerId (int) | Retrieves visit requests associated with a specific prisoner. |
| getPendingVisitRequestsCount() | Public Metrics Method | — | Counts and returns the number of pending visit requests. |
| updateVisitRequestStatus(...) | Public Controller Method | Request ID, status, notes | Updates the approval status and administrative notes of a visit request. |
| getVisitRequestById(int id) | Public Query Method | requestId (int) | Retrieves a visit request using its unique request ID. |
| loadVisitRequestsToTable(JTable table) | Public UI Loader | JTable | Loads all visit requests into a Swing table for administrative review. |

#### 3.3.4 SearchOperation

#### 3.3.4.1 Purpose

Stateless class providing search algorithms (binary search by ID, linear search by name/crime) used by PrisonController

#### 3.3.4.2 Key attributes

* None (stateless; all methods are static).

#### 3.3.4.3 Method Description

Table : Method Description for SearchOperation

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| binarySearchById (LinkedList <PrisonerModel> prisonDetails, int targetId) | Static Search Algorithm Method | prisonDetails (LinkedList <Prisoner Model>),  targetId (int) | Performs a binary search to locate a prisoner by ID. Requires the prisoner list to be sorted in ascending order by ID. Converts the linked list to an array for index-based access, logs step-by-step execution for demonstration, and returns the matching prisoner or null if not found. Time complexity: O(log n). |
| linearSearchBy NameOrCrime( LinkedList <PrisonerModel> prisonDetails,  String searchTerm) | Static Search Algorithm Method | prisonDetails (LinkedList <Prisoner Model>), searchTerm (String) | Performs a linear search over unsorted prisoner data to find partial, case-insensitive matches against prisoner names or crime types. Sequentially scans all records, collects matching prisoners, and logs representative steps for educational clarity. Time complexity: O(n). |
| searchPrisoners (LinkedList <PrisonerModel> prisonDetails, String searchType, String searchTerm) | Static Search Routing Method | prisonDetails (LinkedList <Prisoner Model>), searchType (String), searchTerm (String) | Acts as the central search dispatcher. Validates input, determines the appropriate search algorithm based on the selected search type, invokes either binary or linear search accordingly, and returns a list of matching prisoners. Handles invalid input and runtime errors gracefully. |

### 3.3.5 SimpleQueue

#### 3.3.5.1 Purpose

Minimal FIFO queue implementation used for recently-added items and activity tracking.

#### 3.3.5.2 Key attributes

* private final ArrayList<Object> items — backing store.
* private int head — front index.
* private int rear — last valid index.:

#### 3.3.5.3 Method Description

Table : Method Description for Simple Queue

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| Enqueue  (Object item) | Public Queue Operation | item (Object) | Inserts an element at the rear of the queue following the FIFO (First-In, First-Out) principle. Internally appends the element to the underlying ArrayList and updates the rear index. |
| dequeue() | Public Queue Operation | — | Removes and returns the front element of the queue. Advances the head index instead of shifting elements for efficiency. Automatically resets indices when the queue becomes empty and triggers compaction when required. Throws an exception if the queue is empty. |
| peek() | Public Accessor Method | — | Returns the front element of the queue without removing it. Returns null if the queue is empty. |
| front() | Public Accessor Method (Alias) | — | Provides a semantic alias for peek(), improving code readability when referring to the front of the queue. |
| rear() | Public Accessor Method | — | Returns the rear (last) element in the queue without removing it. Returns null if the queue is empty. |
| isEmpty() | Public State Check Method | — | Checks whether the queue contains any elements by comparing the head index with the current size of the underlying storage. |
| size() | Public Utility Method | — | Returns the number of active elements currently stored in the queue, excluding already dequeued elements. |
| clear() | Public Utility Method | — | Removes all elements from the queue and resets internal indices (head and rear) to their initial states. |
| toArray  (U[] a) | Public Conversion Method | a (Generic Array) | Converts the current contents of the queue into a typed array starting from the front element. Preserves FIFO order and dynamically allocates a new array if the provided one is too small. |
| compactIf  Needed() | Private Memory Optimization Method | — | Reduces memory overhead by compacting the underlying ArrayList when a significant portion of the queue has been dequeued. Copies active elements into a new list and resets internal indices. |
| resetIf  Empty() | Private State Management Method | — | Resets internal storage and indices when the queue becomes empty after dequeue operations, ensuring consistent state and preventing index overflow. |

### 3.3.6 SimpleStack

#### 3.3.6.1 Purpose

Simple LIFO stack used for the trash bin (supports overflow/underflow semantics).

#### 3.3.6.2 Key attributes

* private static final int MAX\_SIZE — stack capacity.
* private final Object[] elements — fixed-size backing array.
* private int top — top index (-1 when empty).

Table : Method description for Simple Stack

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Param-eters** | **Description** |
| Push  (Object item) | Public Stack Operation | Item (Object) | Inserts an element onto the top of the stack following the LIFO (Last-In, First-Out) principle. Checks for stack overflow and throws an exception if the maximum stack size is reached. |
| pop() | Public Stack Operation | — | Removes and returns the top element of the stack. Clears the reference to avoid memory leaks and throws an exception if the stack is empty (stack underflow). |
| peek() | Public Accessor Method | — | Returns the element at the top of the stack without removing it. Returns null for empty stack. |
| top() | Public Accessor Method | — | Provides a semantic alias for peek(), improving code readability when referring to the top element of the stack. |
| isEmpty() | Checks Public State | — | Determines whether the stack contains any elements by checking if the top index is -1. |
| size() | Public Utility Method | — | Returns the number of elements currently stored in the stack. |
| clear() | Public Utility Method | — | Removes all elements from the stack by iteratively clearing references and resetting the top index to its initial empty state. |
| toArray  (U[] a) | Public Conversion Method | A i.e Generic array | Converts the contents of the stack into a typed array in insertion order (bottom to top). Ensures the last element in the array represents the top of the stack and dyna-mically allocates a new array if the provided one is in sufficient. |

### 3.3.7 SortOperation

#### 3.3.7.1 Purpose

Stateless collection of sorting algorithm implementations (Insertion, Selection, Merge) and comparator logic used to sort prisoner lists.

#### 3.3.7.2 Key attributes

* None (stateless; sorting helpers are private static methods).

#### 3.3.7.3 Method Description

Table : Method Description for SortOperation

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| comparePrisoners (PrisonerModel a, PrisonerModel b,  String sortBy, boolean ascending) | Private Comparison Utility Method | a (PrisonerModel), b (PrisonerModel), sortBy (String), ascending (boolean) | Compares two prisoner records based on the selected sorting field and order. Supports multiple attributes including name, ID, age, admission date, release date, and sentence duration. Ensures consistent ordering and handles null values appropriately. |
| selectionSort (LinkedList <PrisonerModel> list,  String sortBy,  boolean ascending) | Private Sorting Algorithm Method | list (LinkedList <PrisonerModel>), sortBy (String), ascending (boolean) | Implements Selection Sort to arrange prisoner records based on the specified field and order. Iteratively selects the best candidate and swaps it into position. Intended for educational clarity. Time complexity: O(n²). |
| insertionSort (LinkedList <PrisonerModel> list, String sortBy, boolean ascending) | Private Sorting Algorithm Method | list (LinkedList <Prisoner Model>), sortBy (String), ascending (boolean) | Implements Insertion Sort to arrange prisoner records while preserving stability. Efficient for small datasets and nearly sorted lists. Demonstrates element shifting logic. Time complexity: O(n²). |
| mergeSort( LinkedList <PrisonerModel> list,  String sortBy, boolean ascending) | Private Sorting Algorithm Method | list (LinkedLis t<Prisoner Model>), sortBy (String), ascending (boolean) | Implements Merge Sort using divide-and-conquer strategy. Recursively splits the list, sorts sublists, and merges them back while maintaining order stability. Suitable for large datasets. Time complexity: O(n log n). |
| sortPrisoners (LinkedList <PrisonerModel> prisonDetails, String sortBy, boolean ascending) | Public Sorting Entry Method | prisonDetails (LinkedList <Prisoner Model>), sortBy (String), ascending (boolean) | Provides a simplified entry point for sorting prisoner records. Automatically selects an appropriate sorting algorithm based on dataset size and returns a new sorted list without modifying the original. |
| sortPrisoners (LinkedList <PrisonerModel>  prisonDetails, String sortBy, boolean ascending, String algorithm) | Public Sorting Entry Method (Overload) | prisonDetails (LinkedList <Prisoner Model>), sortBy (String), ascending (boolean), algorithm (String) | Allows explicit selection of the sorting algorithm from the UI. Routes the request to the internal sorting logic while preserving consistent behavior and validation. |
| sortInternal (LinkedList <PrisonerModel> prisonDetails, String sortBy, boolean ascending, String algorithm) | Private Sorting Coordinator Method | prisonDetails (LinkedList <Prisone rModel>), sortBy (String), ascending (boolean), algorithm (String) | Centralizes sorting logic. Validates input data, determines the most suitable sorting algorithm when none is specified, executes the sorting process, logs execution details, and handles user feedback and error recovery. |

### 3.3.8 TrashBinOperation

#### 3.3.8.1 Purpose

Stateless helpers to push/pop/inspect/empty the trash SimpleStack and present user-facing messages for restore/permanent delete flows.

#### 3.3.8.2 Key attributes

* None (stateless; operations accept the SimpleStack as parameter).

#### 3.3.8.3 Method Description

Table : Method Description for TrashBinOperation

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Type** | **Key Parameters** | **Description** |
| pushToTrash (SimpleStack trashBin, PrisonerModel deletedPrisoner) | Public Stack Operation Method | trashBin (SimpleStack), deletedPrisoner (PrisonerModel) | Pushes a deleted prisoner onto the trash bin stack, demonstrating the stack push operation. Implements LIFO behavior so the most recently deleted prisoner will be restored first. Logs the operation for educational clarity. |
| popFromTrash (SimpleStack trashBin, LinkedList <PrisonerModel> prisonDetails) | Public Restore Operation Method | trashBin (SimpleStack), prisonDetails (LinkedList <PrisonerModel>) | Restores the most recently deleted prisoner from the trash bin using the stack pop operation. Prompts for user confirmation, removes the prisoner from the stack, reinserts them into the active prisoner list, and provides UI feedback. Returns the restored prisoner or null if restoration is cancelled or the trash is empty. |
| viewTrashContents (SimpleStack trashBin) | Public Read-Only Display Method | trashBin (SimpleStack) | Generates an HTML-formatted view of the trash bin contents without modifying the stack. Demonstrates stack **peek** and traversal while clearly showing LIFO order, including identification of the top element. |
| getTrashSize (SimpleStack trashBin) | Public Utility Method | trashBin (SimpleStack) | Returns the number of prisoners currently stored in the trash bin stack. |
| isTrashEmpty (SimpleStack trashBin) | Public State Check Method | trashBin (SimpleStack) | Checks whether the trash bin stack is empty, allowing the UI to enable or disable restore-related actions. |
| emptyTrash (SimpleStack trashBin) | Public Destructive Operation Method | trashBin (SimpleStack) | Permanently clears all prisoners from the trash bin stack after user confirmation. Demonstrates the stack **clear** operation and enforces irreversible deletion behavior. |

# Wireframe Design and User Interface Screens

## 4.1 Landing Page

The Nepal Correctional Facilities landing page of the Prison Management System has a clear, functional design with the emphasis made on the visibility of data and the simplicity of navigation. The interface has been designed to have three different horizontal levels with a light cyan background, which makes the environment highly contrasted and allows fast information processing. Going up the page, there is a large blue header which is the main branding spot, showing the title of the system and that it is specifically applied to the digital correctional infrastructure of Nepal in a clear and non-serfaced font.

Just below the header, the UI has a data-driven dashboard with four white rectangular cards which are equal and uniform. These cards post very significant institutional statistics: the number of prisoners in total, the existing occupancy rate (0.28 percent), and the demographic composition of the population by gender. This is because of this centralized positioning of the statistics that made sure that the administrators had instant access to high level situational awareness as soon as they opened the page.

User authentication and role-based access is allocated to the bottom segment of the interface. It has two big color-coded buttons, which differentiate between internal and external stakeholders. There is a purple button labeled as Admin Login that enables authorized individuals to access the management backend, and a bright green button labeled as Family Portal that has a special access point to the general population and close family of the incarcerated. This visual isolation of functions reduces friction in the hands of the users and increases the functionality of the digital management platform.



Figure : Wireframe of Landing Page



Figure : Actual Landing Page

## 4.2 Admin Login Page

The Admin Login interface for the Prison Management System maintains the established visual identity while transitioning to a specialized functional state. The page is anchored by a bright cyan header that prominently displays the title "Admin Login," ensuring the user’s current location within the system is immediately clear. To enhance navigation efficiency, a "Back to Home" button is situated in the top-left corner of the header, providing a simple exit path for users who may have navigated to this secure area in error.

The core of the interface features a centrally aligned, white authentication module that contrasts sharply against the light cyan background. This module is titled "Admin Access" and includes a concise instructional subtitle, "Enter your credentials to continue," which guides the user through the security process. The form itself follows a standard vertical layout, utilizing labeled input fields for "Username" and "Password" to ensure clarity.

Completion of the authentication process is facilitated by a blue "Login" button at the base of the module, which serves as the primary call-to-action for administrative personnel seeking system entry. This layout adheres to modern UI design principles by minimizing cognitive load and focusing the user's attention entirely on the secure login task. The overall design ensures a seamless transition from the public-facing dashboard to the restricted administrative environment.

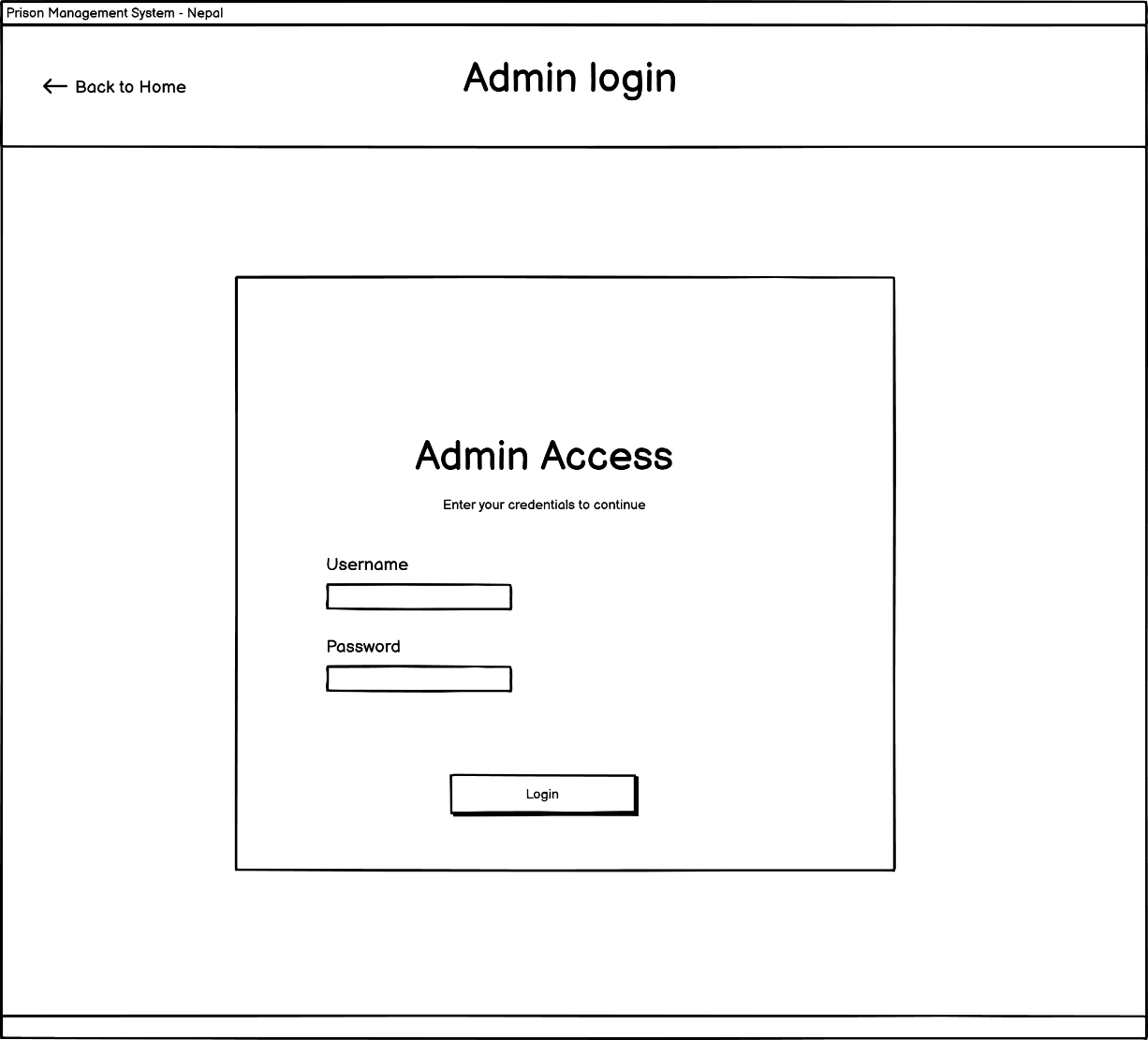


Figure : Wireframe of Admin Login page

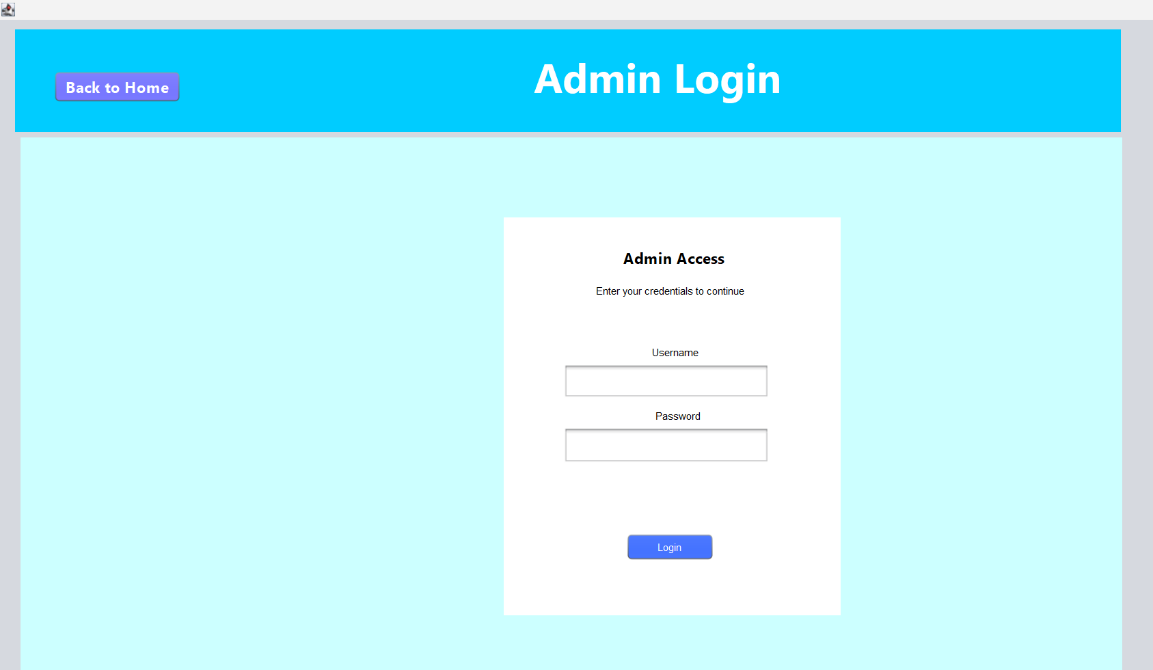


Figure : Actual Admin login page

## 4.3 Admin Dashboard Page

The Nepal Correctional Facilities Admin Dashboard interface is the main data management center, and it is based on a dense and information-dense layout to enable administrative control. The page is crowned with a vivid blue header that recognizes the section as the "Admin Dashboard" and a red "LogOut" button that has a high visibility level in the upper-right hand corner to end the secure session safely. Under the header, a detailed control panel will give administrators a range of data manipulation tools, including a search box with a linear search algorithm option, sorting options (such as Admission Date and InsertionSort), and main action buttons, such as "Add Prisoner," "Visit Requests" and "Trash Bin."

A detailed table of the prisoner records is the main element of the dashboard, and it presents a tabular representation of the population of inmates. The critical data required in every entry are ID, Name, Age, Gender, Admission Date, Crime Type, Release Date, Sentence duration, Location and Status. To further streamline the handling of individual records, the right column of the table contains local action buttons, in this case, View, Edit, and Delete, which enable an individual to have a very specific control at the specific entry of inmate records without literally leaving to the main list.

The right side of the interface, complementary to the central table, is a "Recent Activities" log, which is a real time chronological audit trail of the system changes. Recent additions to the database are shown in this sidebar in the form of names and the ID of the newly registered inmates and the actual timestamps. With the integration of high-end control mechanisms, an elaborate central database, and a live activity feed, the dashboard enables a solid platform to control the complex correctional data in a transparent and efficient manner.

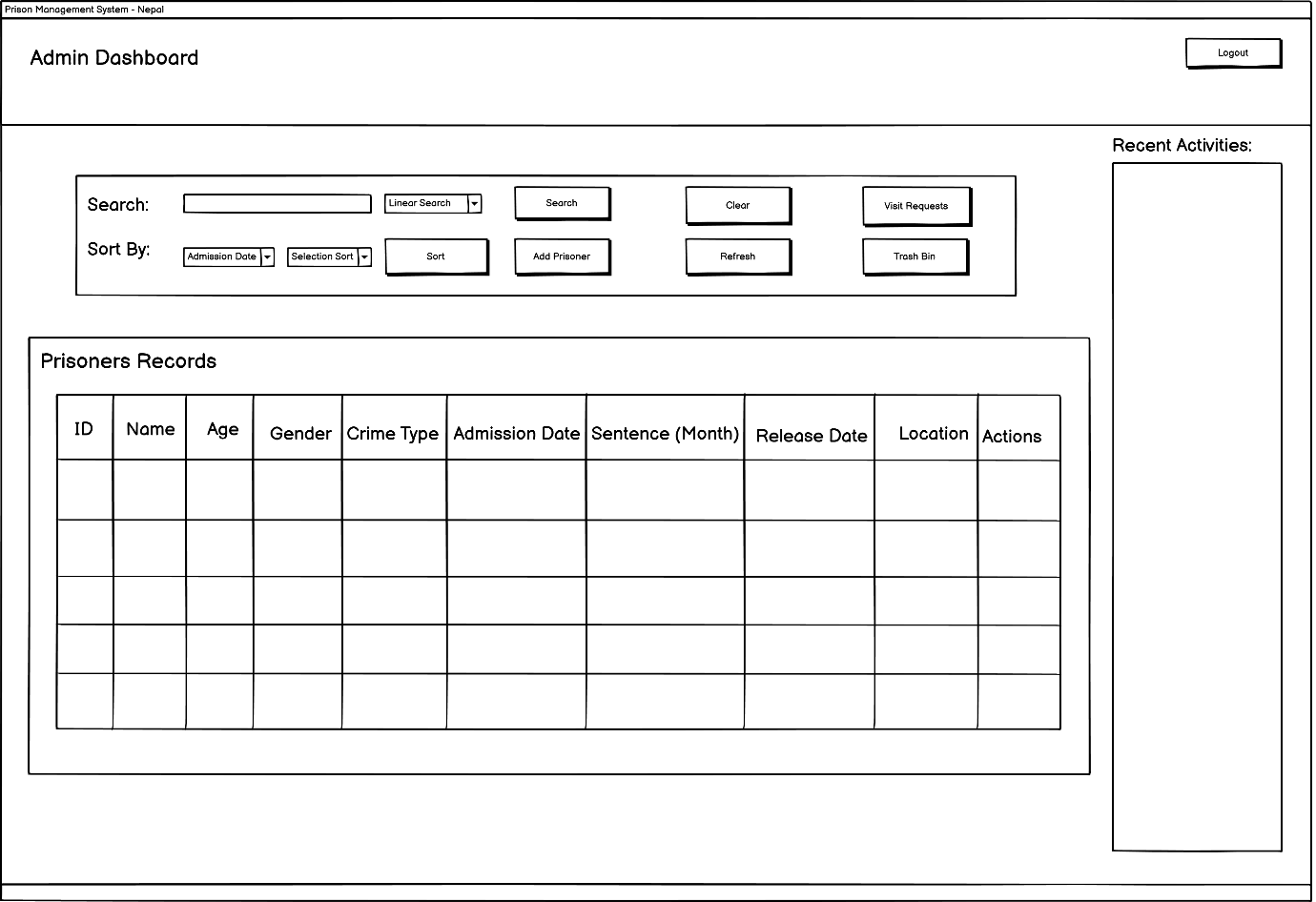


Figure : Wireframe of Admin Dashboard

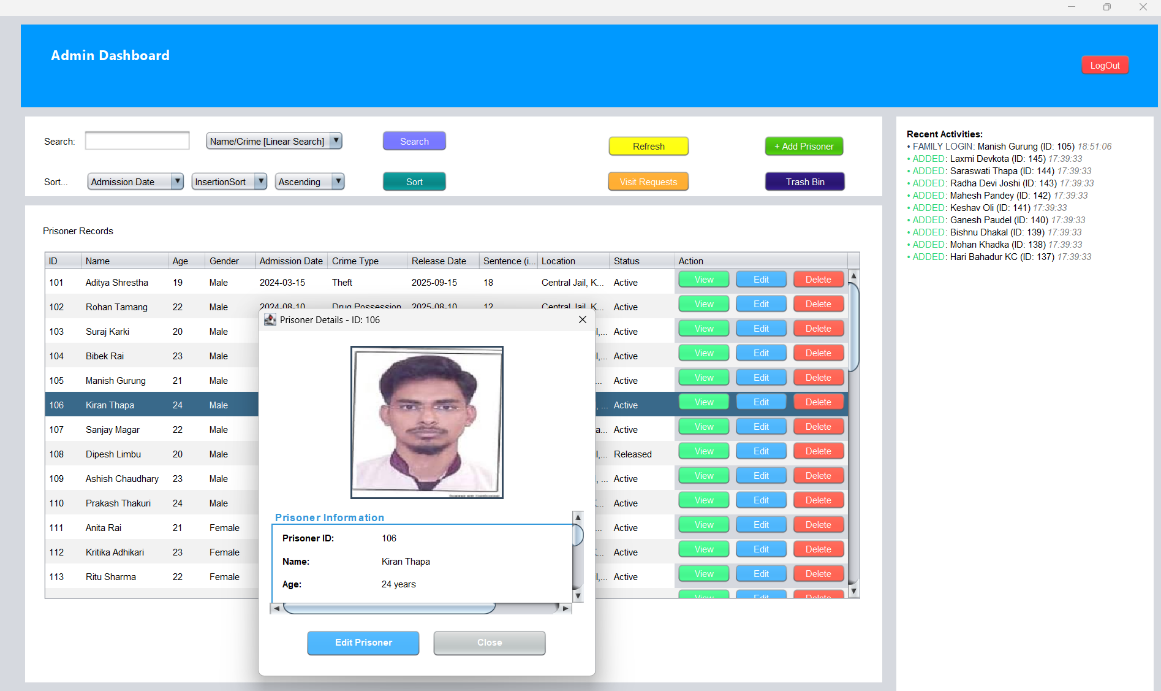


Figure : Actual Admin Login Dashboard

## 4.4 Family Login Page

The Nepal Correctional Facilities Family login Portal is user-centric in the design, and the use of a unique color profile ensures that it is distinctly differentiated in the administrative environment. A vibrant lime-green header with the title of the Family Login Portal and a navigation button of Back to Home defines the interface, which makes it predictable that the user will be guaranteed a uniform user experience of returning to the main landing page. This white header is contrasted with a light-grey background, making the user draw his/her attention to the major authentication box.

The key module, which is called Family Portal Access, has a supportive subtitle, namely, to connect with your loved ones, to create a supportive atmosphere of the end-user. An instructional box on how to access is highly delivered as a bright colored box above the fields of input and the necessary credentials are outlined, namely, a prisoner ID offered by the prison and a registered family code. This simplicity of included instructions helps to decreases user anxiety and technical jostling when it comes to the process of logging-in.

The authentication form should work in two ways, where a Prisoner ID and a Family Code are entered and the fields have a text put as a guide on the accuracy of the data in the placeholders. The last call-to-action is a green button at the bottom of the white module that reads an Access Portal, which adheres to the color scheme in the header to make the portal look like itself. Such a minimalistic design allows families to use the system effectively and follow the security measures of the correctional facility.

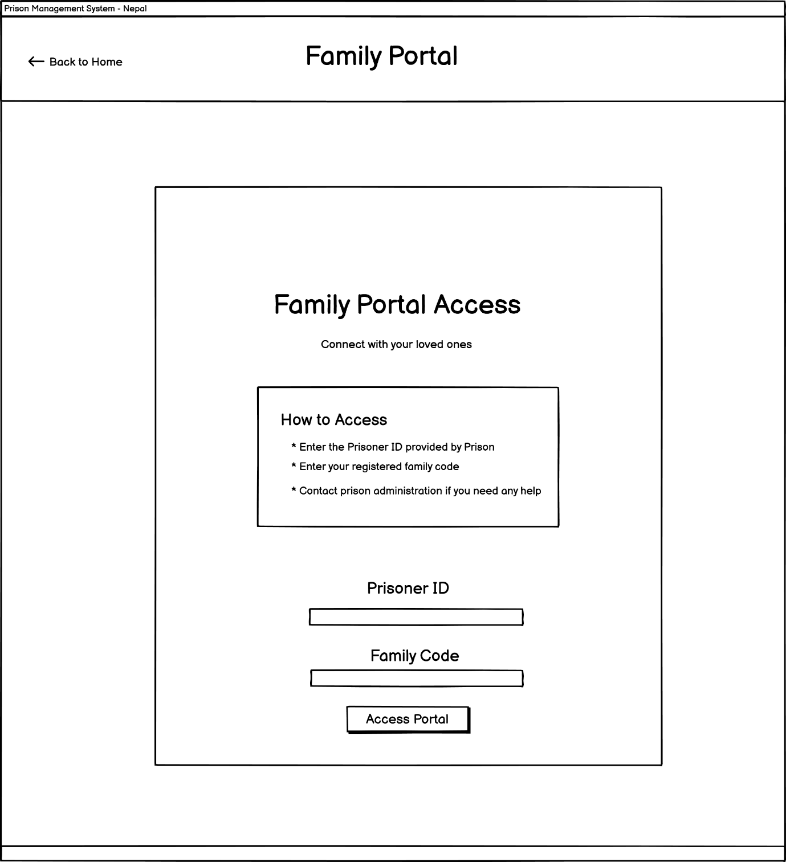
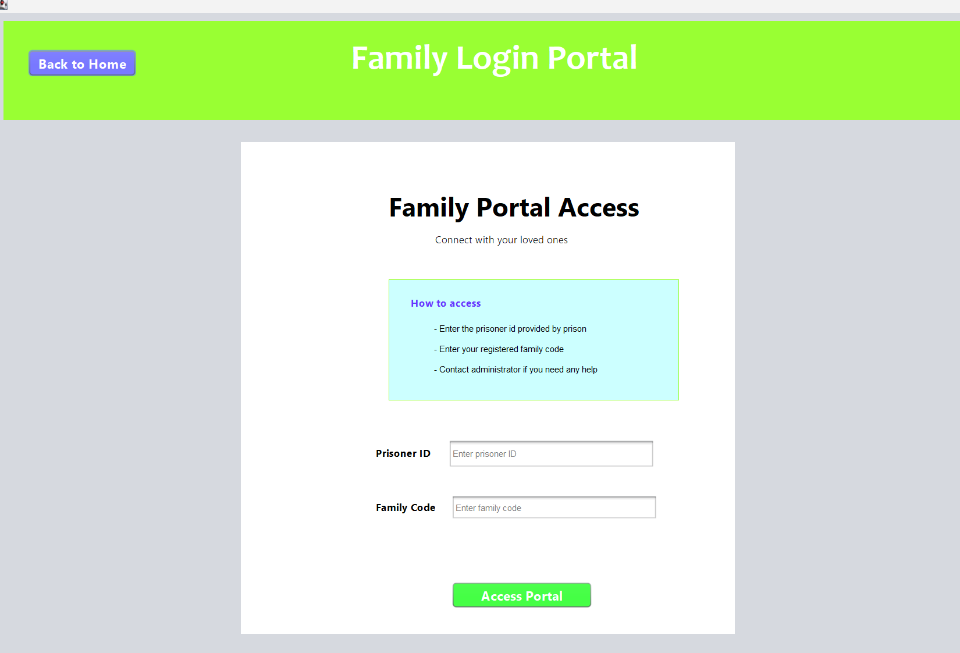


Figure : Wireframe of Family login portal



## 4.5 Family Dashboard Page

The Family Portal Dashboard becomes the main encounter with the family members of the authorized relatives, where the lime-green and grey feel is carried on to the login phase to create a feeling of visual continuity. The interface has a visible green header on the top that displays the location the user is in the "Family Portal Dashboard" and a purple log out button to manage the secure session. This is the highest level of navigation that has made sure that the user can access the purpose of the system easily with a straightforward way out.

The design is subdivided into three functional modules which focuses on transparency and communication between the family and the facility. The highest module is the Prisoner Information where the information is condensed into a profile of the incarcerated and that includes the name, their current health status, date of admission to the prison, the location of the prison and the expected date of release. Such an element of data sharing plays a vital role in helping families to be at ease about the status and welfare of their relatives.

The bottom portion of the dashboard is devoted to the visitation process with a form of a requesting a visit and log of the history of visits. The request section enables the families to place fresh visits by filling in their name, relations with the prisoner, a favored date, and the visit purpose. At the bottom of this is a table history showing past and future visits, and their status (i.e. pending) where the family can be informed of administrative approvals. Such a combined solution simplifies the logistical aspects of prison visits, which creates a more convenient interaction between prisoners and their support networks.

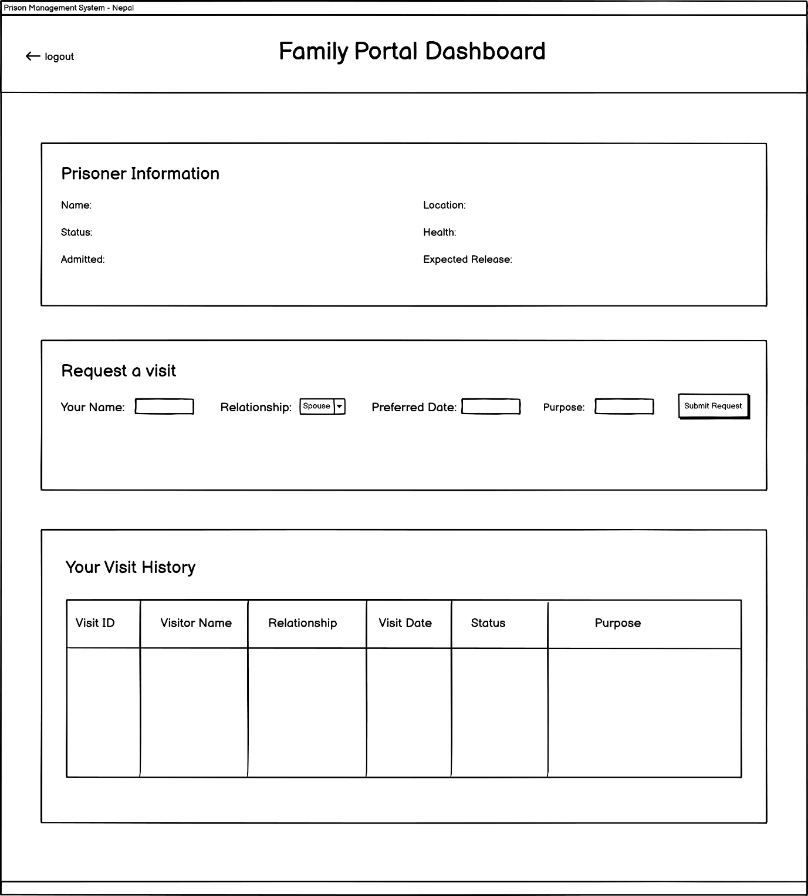
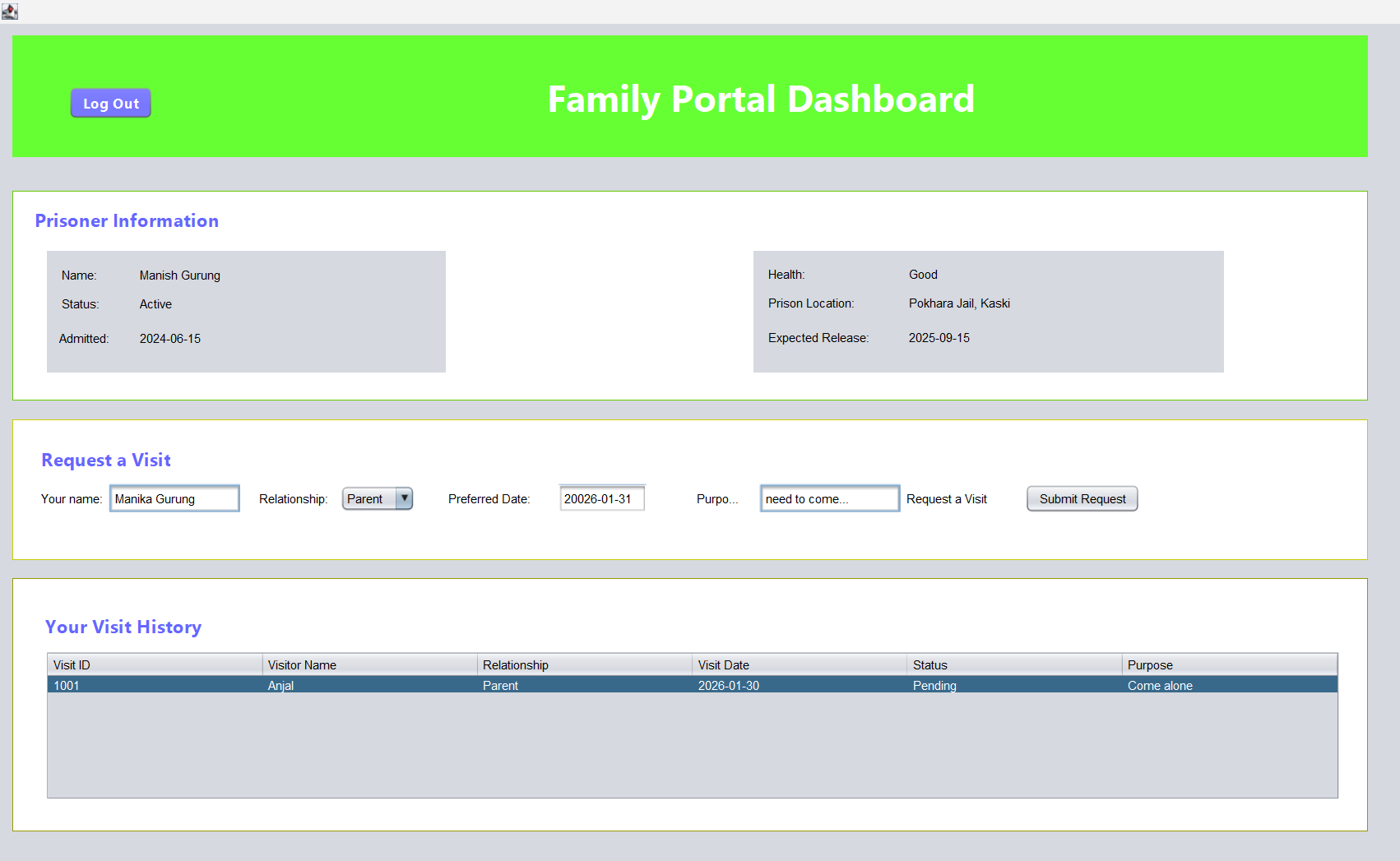


Figure : Wireframe of Family Dashboardp



# Coursework Developement

## Tools Used

### Apache Netbeans

**Tool Usage**

**Evidence**

## CRUD Implementation

## Algorithms Implemented

Searching Algorithms

Sorting Algorithms

# CRUD Functionality

## 5.1 Create Functionality

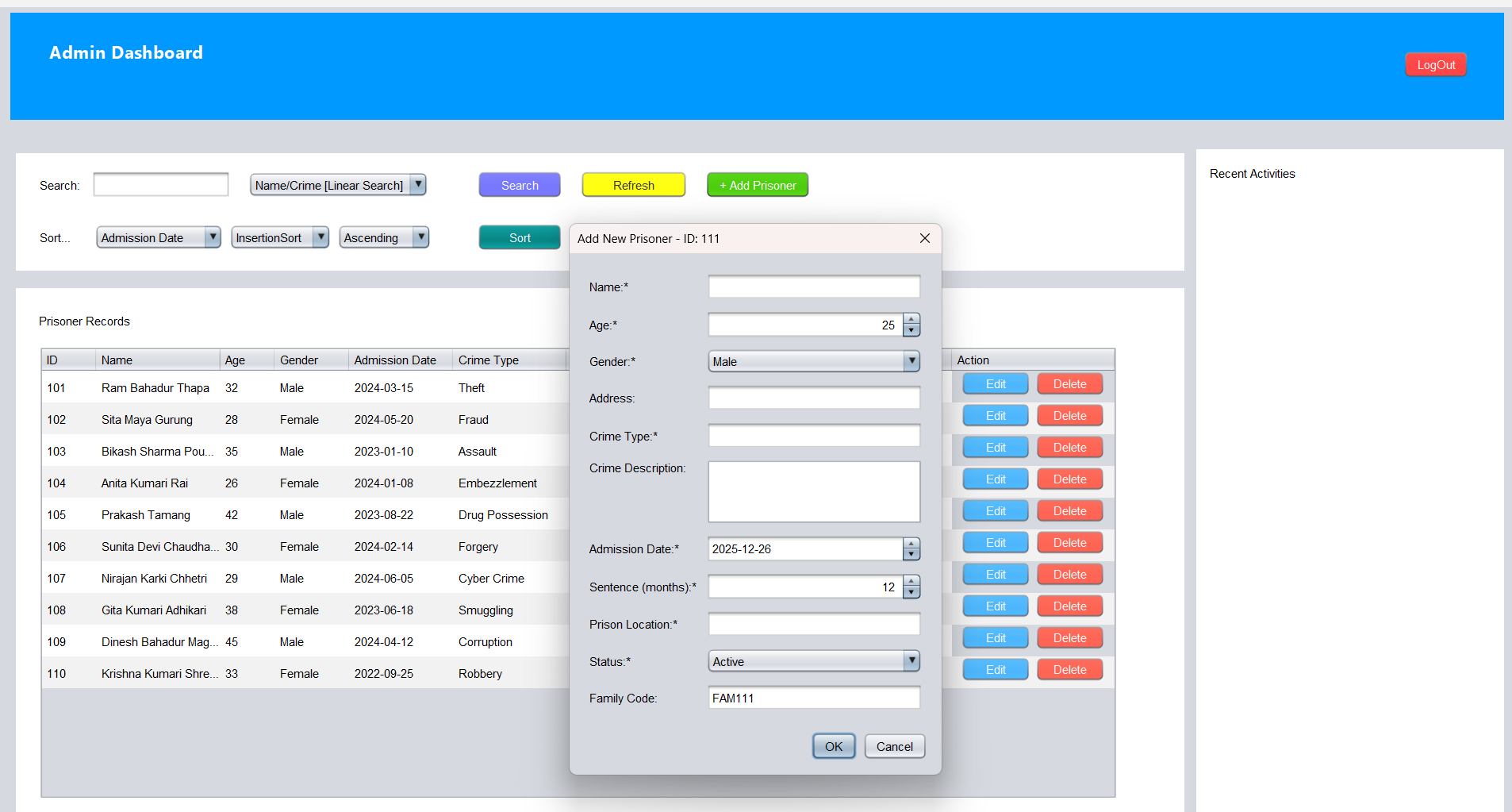


Figure : Dialog Box to provide new prisoner detials

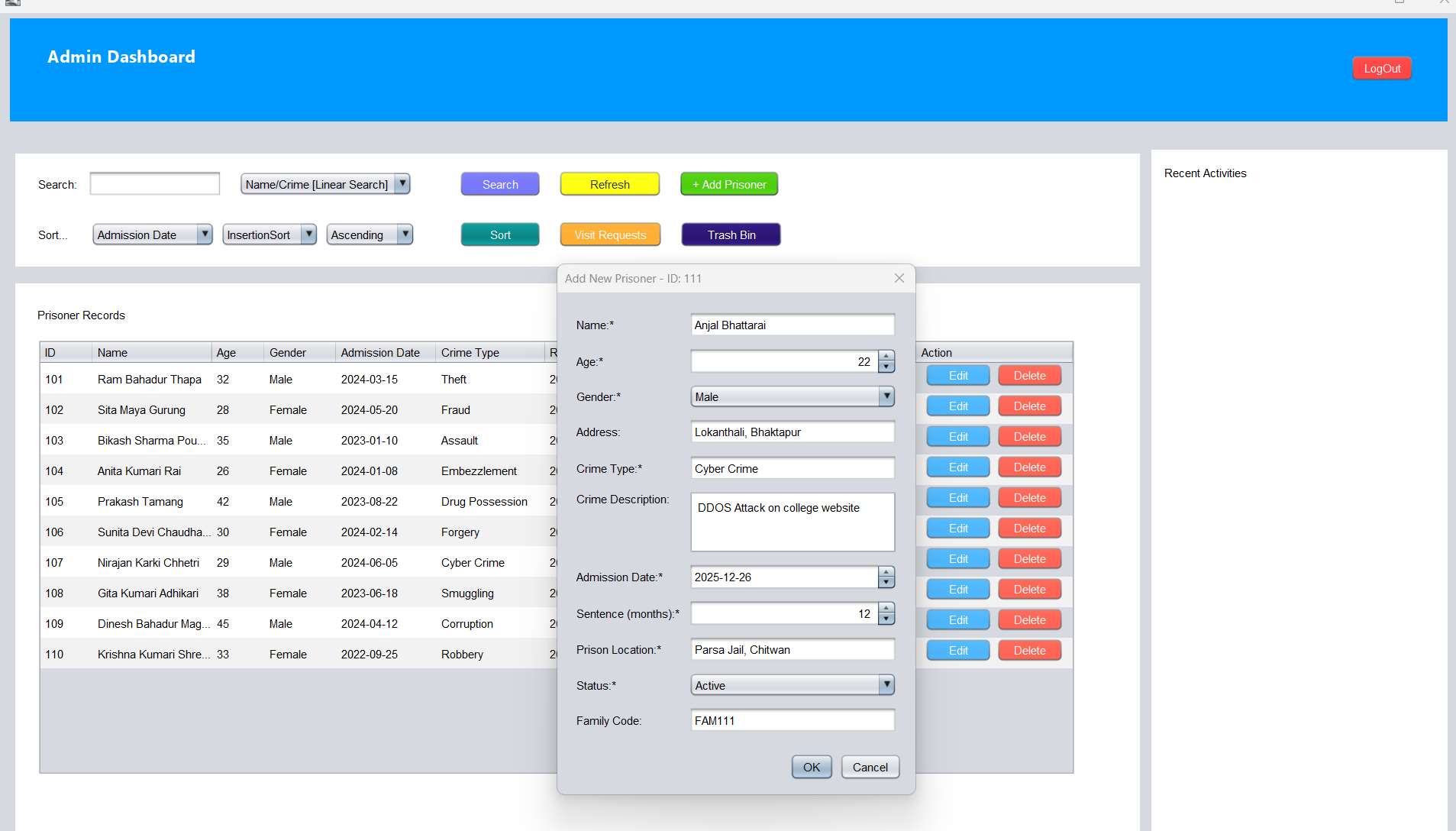


Figure : New Prisoner being added

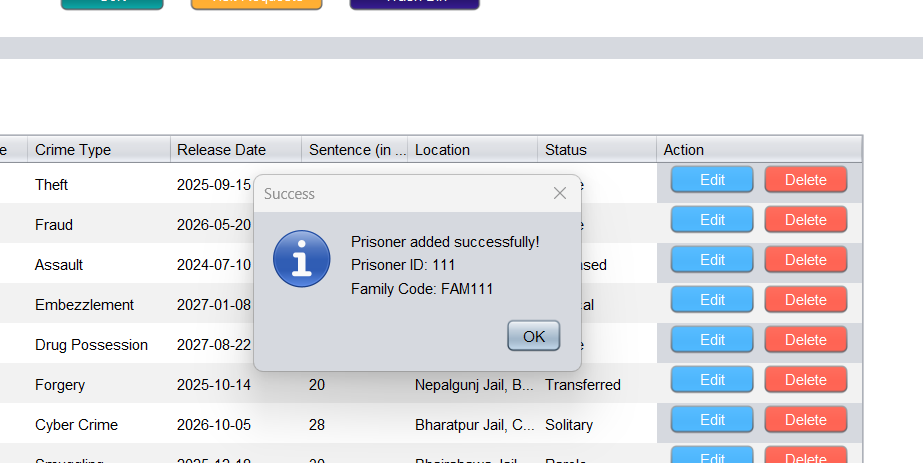


Figure : Create new Prisoner Successfully

## 5.2 Read Functionality

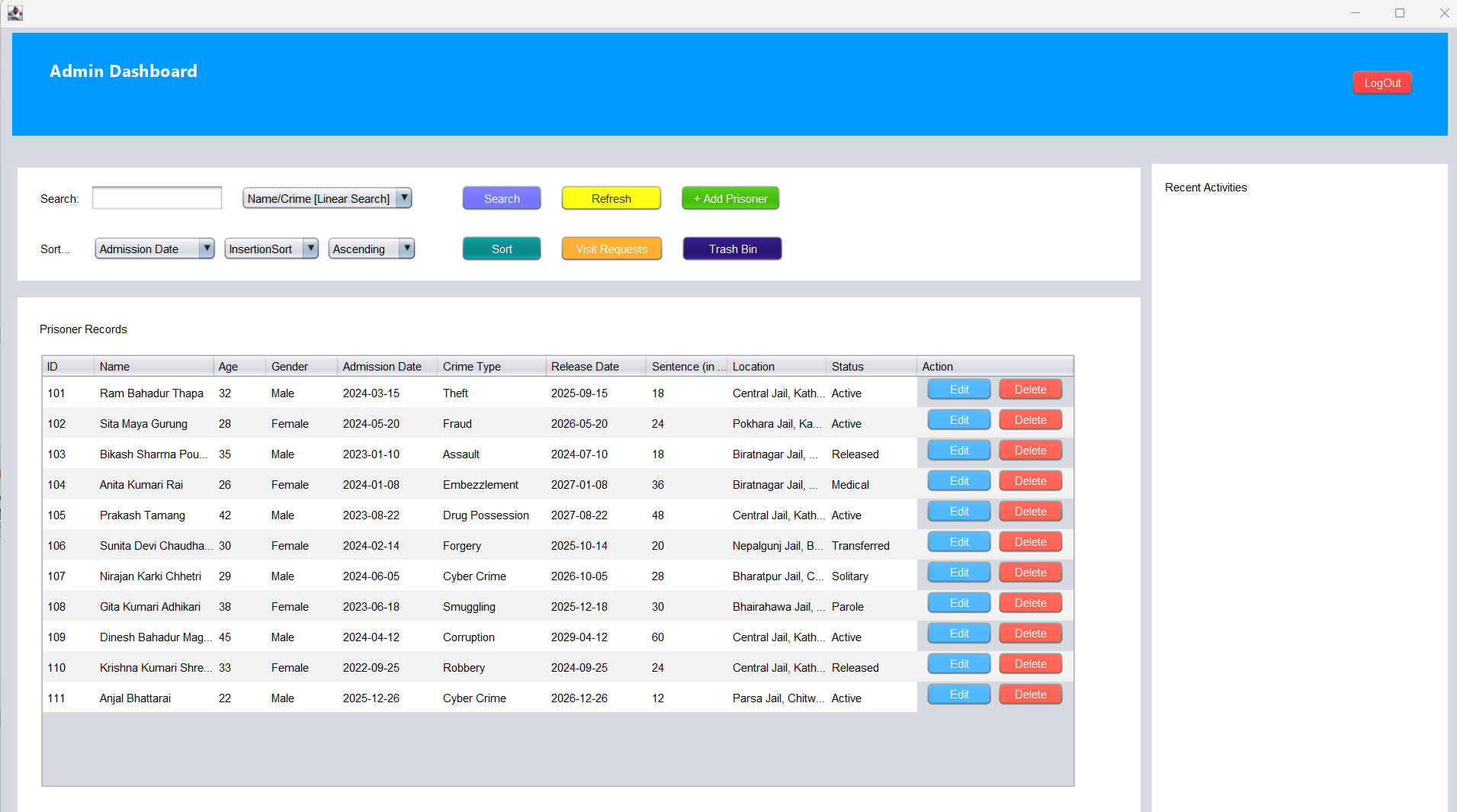


Figure : Detail over view of all Prisoner

## 5.3 Update Functionality

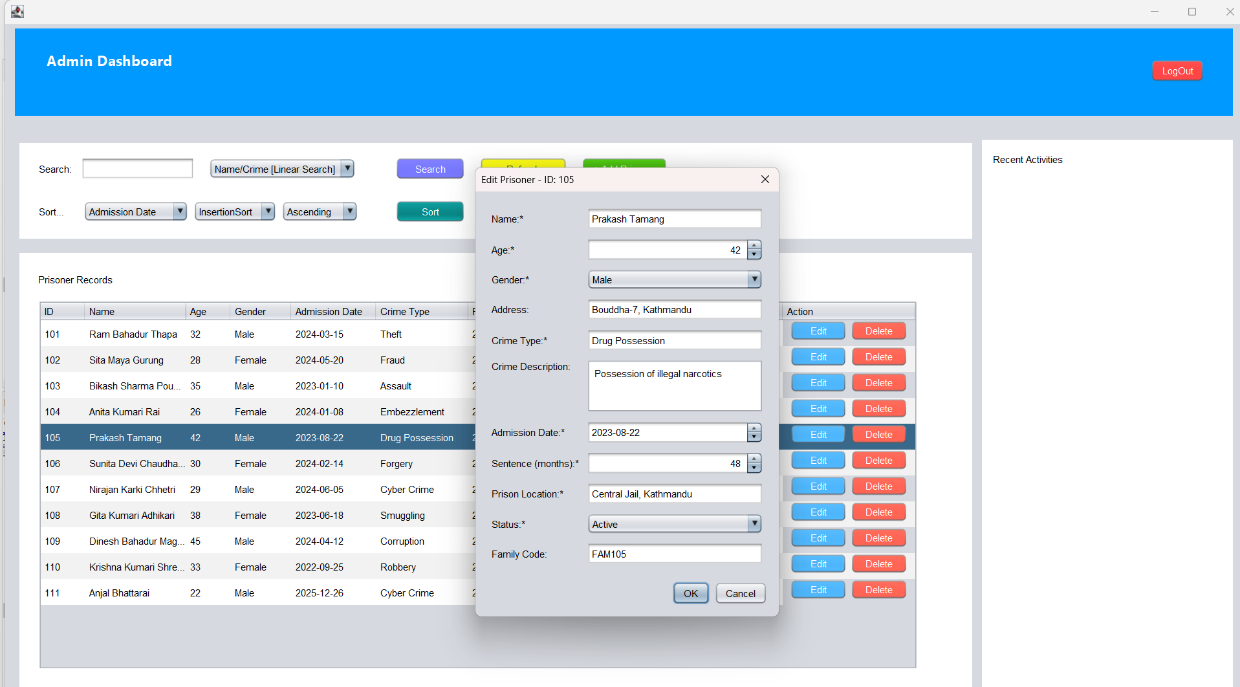


Figure : Editable current details of prisoner when clicked edit button

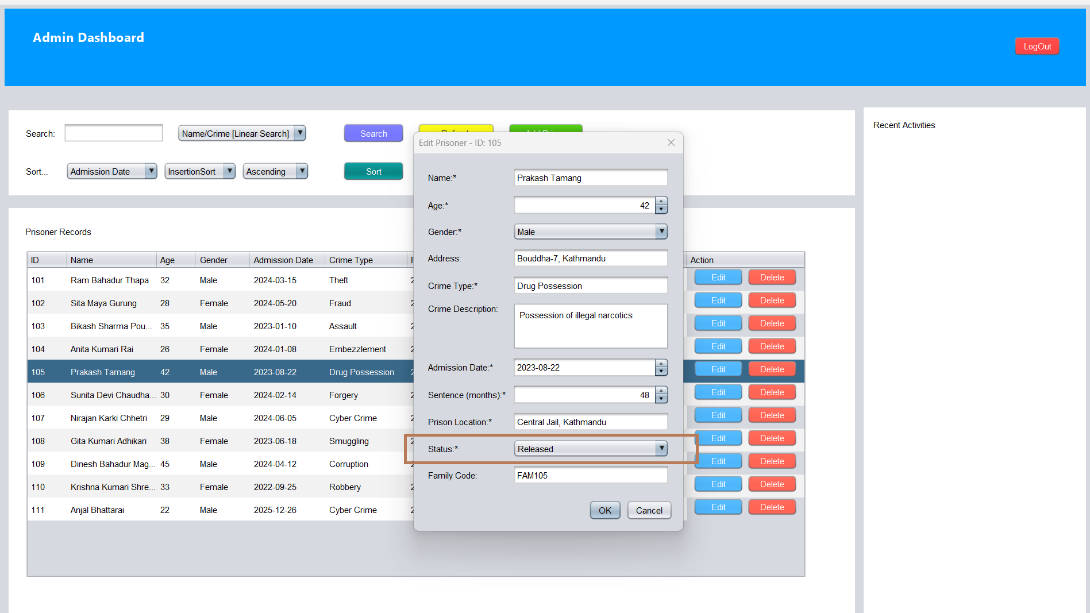


Figure : Status changed from active to released

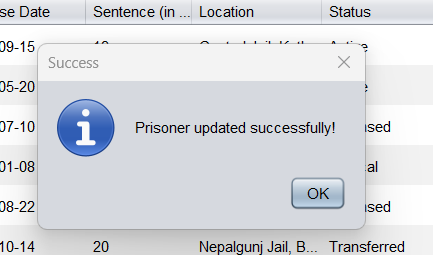




Figure : Updated details of prisoners

## 5.4 Delete Functionality

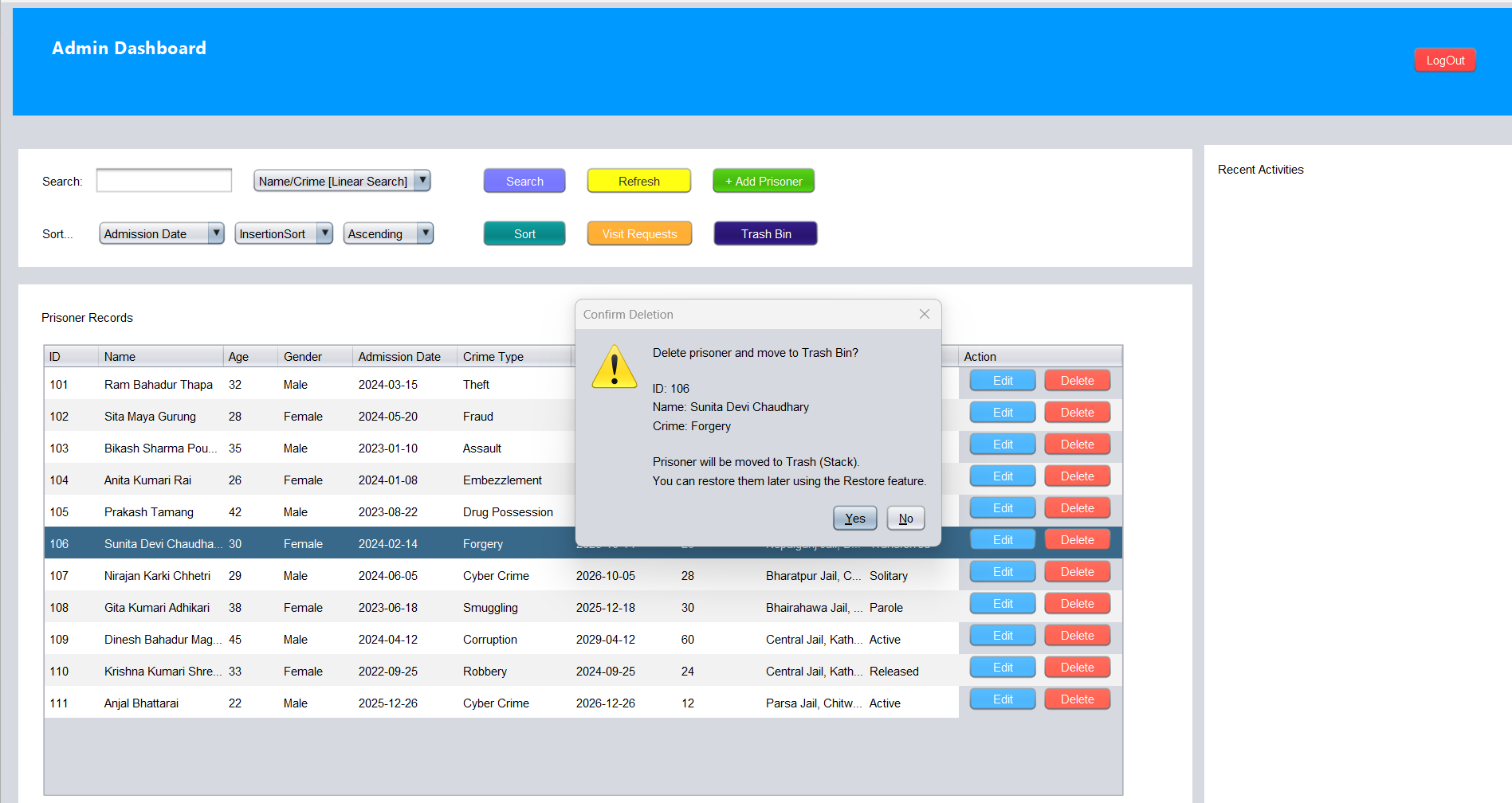


Figure : DIalog Box to confirm deletion of prisoner details

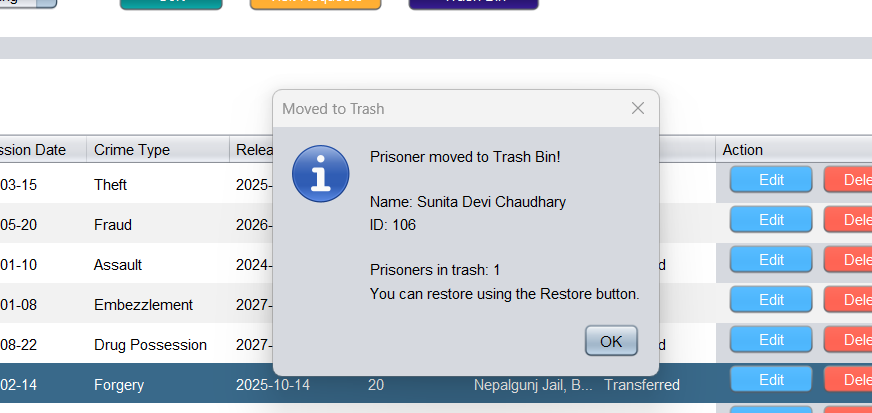


Figure : Deletion Confirmation Pop-up

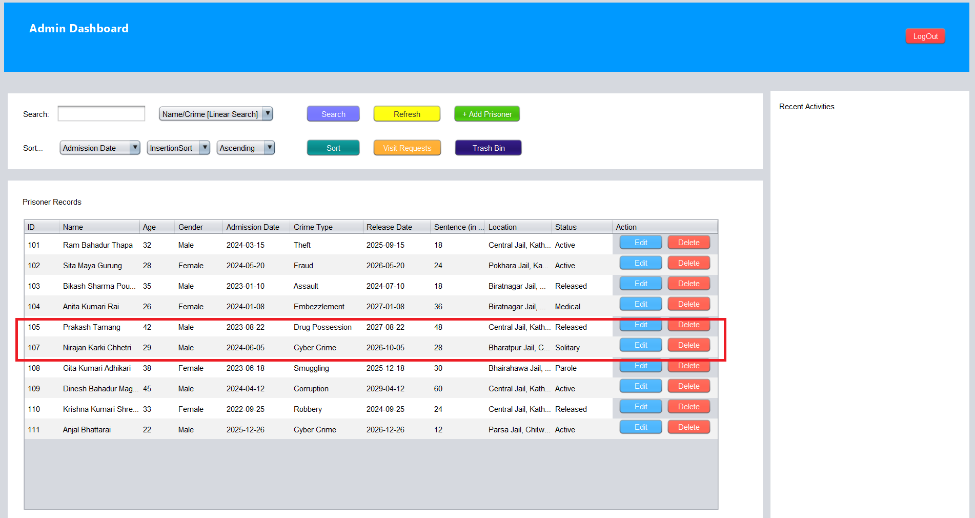


Figure : Prisoner with id 106 deleted

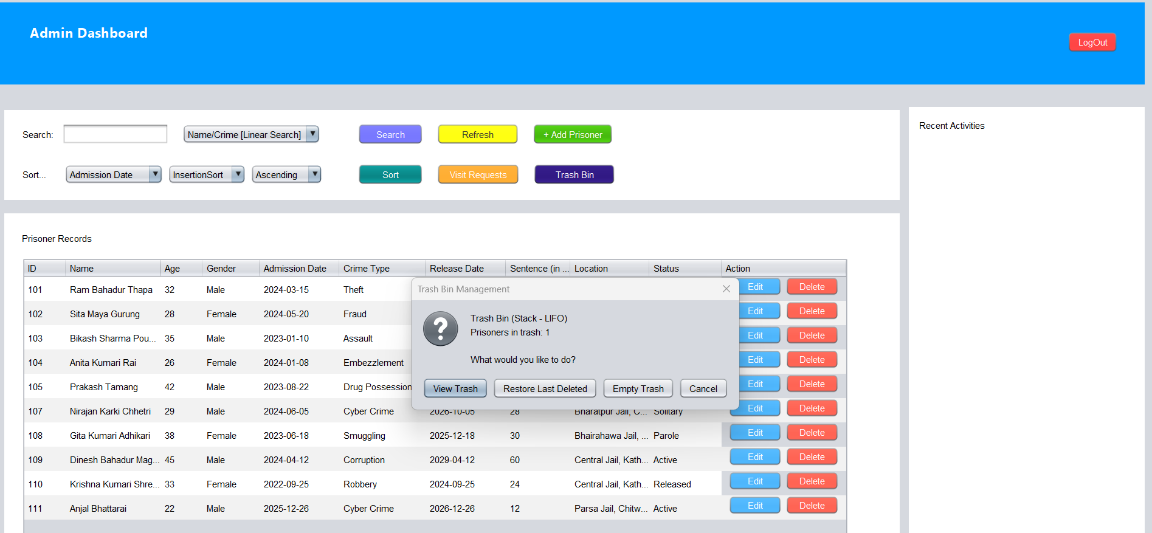


Figure : Recently deleted prisoner stored in STACK

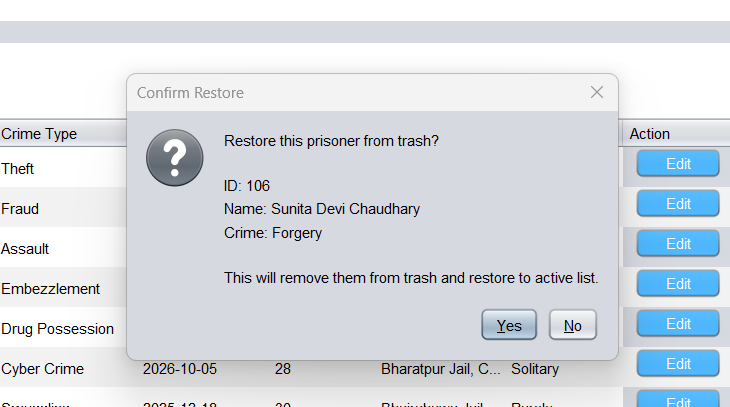


Figure : Restore Confirmation

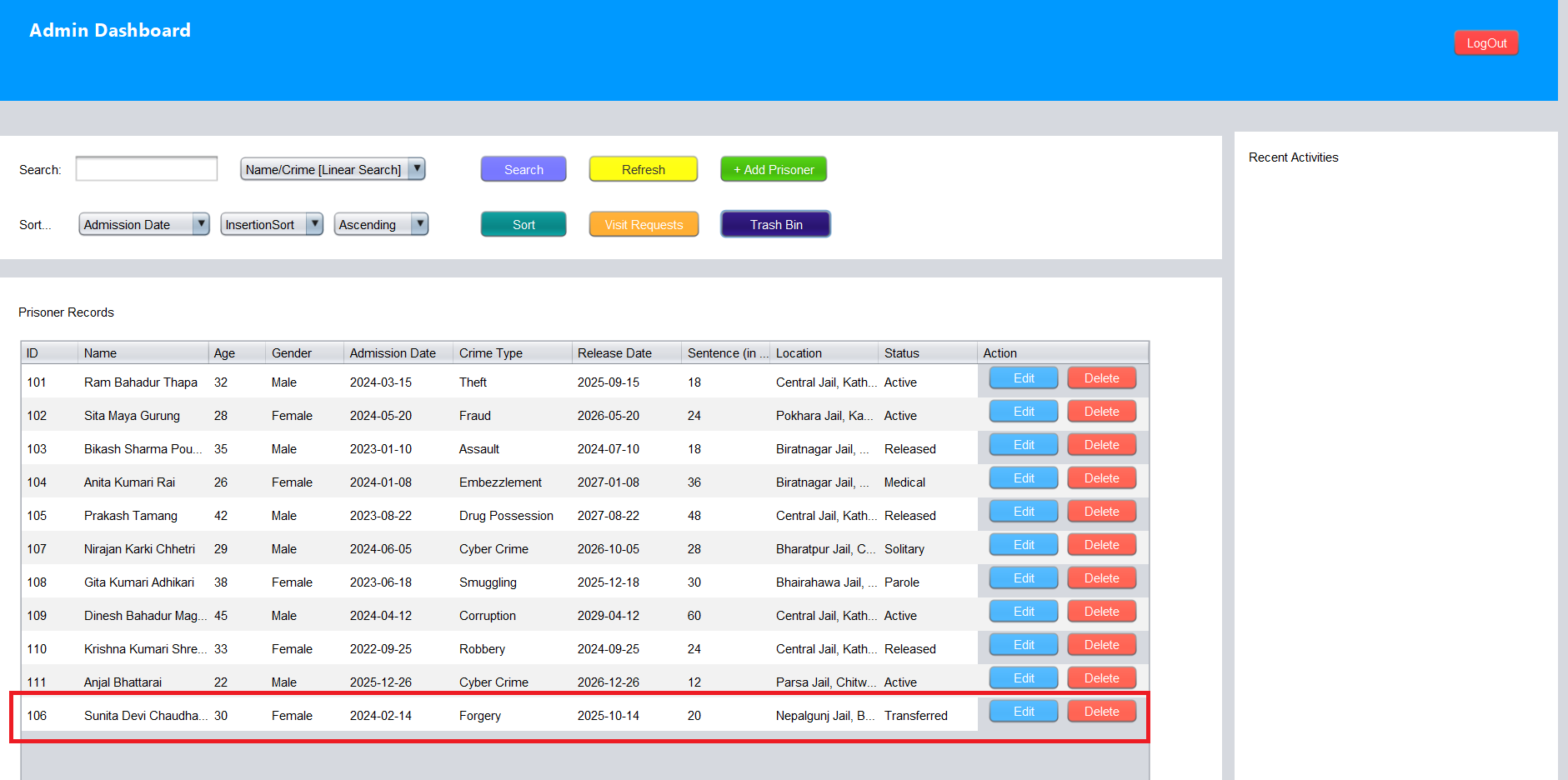


Figure : Restored Deleted Prisoner and Stored back in QUEUE

# 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 : Add new prisoner with empty name    Figure : 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 : 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 : Add Prisoner with invalide Date format    Figure : 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