Final Year Design Project

Prison Management System: Optimizing Inmate Record sand Facility Work-flow

By

Ahmad Zaman BCSF21M522

Rehmat Amjad BCSF21M524

Khadija Hafeez BCSF21M520

Sana Shahid BCSF21M541

Under the supervision of

Prof. Umair Babar

Bachelor of Science in Computer Science (2021-2025)

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE.

Prison Management System: Optimizing Inmate Record sand Facility Work-flow

A project presented to

University of the Punjab, Lahore

In partial fulfillment

of the requirement for the degree of

Bachelors of Science in Computer Science (2021-2025)

By

Ahmad Zaman BCSF21M522

Rehmat Amjad BCSF21M524

Khadija Hafeez BCSF21M520

Sana Shahid BCSF21M541

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE

DECLARATION

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other, we will stand by the consequences. No portion of the work presented has been submitted of any application for any other degree or qualification of this or any other university or institute of learning.

Signature: ----------

Ahmad Zaman [BCSF21M522]

Signature: ----------

KhadijaHafeez[BCSF21M520]

Signature: ----------

Rehmat Amjad [BCSF21M524]

Signature: ----------

Sana Shahid[BCSF21M541]

CERTIFICATE OF APPROVAL

It is to certify that the final year design project (FYDP) of BSCS "Prison Management System: Optimizing Inmate Records and Facility Work Flow" was developed by **AHMAD ZAMAN (BCSF21M522), REHMAT AMJAD (BCSF21M524), KHADIJA HAFEEZ (BCSF21M520),** and **SANA SHAHID (BCSF21M541)** under the supervision of "PROF. UMAIR BABAR " in my opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Computer Science.

Signature:----------.

FYDP Supervisor:

Signatures (Faculty Advisory Committee (FAC)

Signatures			
Name	Mr. Ejaz Ashraf	Mr. Asim Rasul	
	FAC1	FAC2	FAC3

Signature:----------.

Head of FYDP Coordination Office:

Signature:---------- Dated: _____

Chairperson, Department of Computer Science

Executive Summary

The Prison Management System (PMS) project is considered complete and therefore stands out as the modern improvement and automation of operations in correctional facilities, especially in developing countries like Pakistan where manual paper systems are still running the operations. Such very old-age methods produce poor bookkeeping, probably data loss, delay in operations, and much security threats. This project was conceived to be part of the solutions to these problems by the design of a centralized, modular, and secure web application to manage all activities of prisoner data, visitor appointments, healthcare requests, duty schedules of staff, and complaints resolution-all at one system. Its overall objectives were reducing 90% manual paperwork, tracking inmate healthcare better, taking down time and effort spent on visitor scheduling, and providing secure data with role-based access control (RBAC) and modern encryption standards. The platform also provides dashboards for administrators, real-time monitoring tools, and structured workflows for each of the stakeholders, including inmates, staff, the visitor, and the prison administrators. It targets inefficiencies to expedite modernisation of institutional processes thereby increasing operational efficiency and welfare for prisoners plus providing greater transparency and security within the correctional environment.

The method used for the project was a combination of old methodologies of the Software Development Life Cycle (SDLC) with a huge amount of agileness in iterations so that changes of requirements could be easily incorporated. The very first phase captured a large number of requirements through user interviews, domain research, and analyses of existing systems in prisons. System architecture was designed using UML diagrams, which included use case diagrams, sequence diagrams, and class models . The next phase included the preparation of high-fidelity wireframes in Figma with usability and accessibility (WCAG 2.1 compliance) confirmed. The backend development used Django 5.1.1 for the application engine, together with Django REST Framework for API layers. Frontend relied on ReactJS 18.3 for creating a responsive, component-driven user interface. The relational database was ensured by MySQL, which inclined towards data consistency and scalability. CSS advanced practices and Bootstrap 5.3 occupied the front end styling. Major technical content includes real-time dashboards, encrypted communication using TLS 1.3, AES-256-based data storage encryption, and secured role management. Testing was carried out over several user groups and devices, showing almost 99.9% system uptime and 2-3 seconds response times for visitor requests processing. Docker-based deployment, entire system documentation, SRS, SDS and FYP Report, and training modules for user onboarding completed the project. The final product offers scalable potential for becoming a comprehensive digital transformation solution for correctional institutions, thereby streamlining rehabilitation processes, structuring administrative oversight, and generally providing safer prison environments.

Keywords: Prison Management System, Correctional Facility Automation, Inmate Healthcare Management, Visitor Scheduling, Secure Administrative Software.

Acknowledgement

The acknowledgment section is an opportunity to express gratitude and appreciation to individuals and organizations who have contributed to the completion of your Final Year Project (FYP).

Signature: ----------

Ahmad Zaman [BCSF21M522]

Signature:--- ----------

Khadija Hafeez [BCSF21M520]

Signature: ----------

Rehmat Amjad [BCSF21M524]

Signature: ----------

Sana Shahid [BCSF21M541]

Abbreviations

Table of all the abbreviations and acronyms used in your FYP along with their respective brief description. The abbreviation list must be ordered alphabetically.

API	Application Programming Interface: A set of routines and tools for building software applications.
DB	Database: A structured collection of data stored electronically.
FCIT	Faculty of Computing and Information Technology: Academic division focused on computing education.
FYP	Final Year Project: A culminating academic project for degree completion.
HTTP	Hyper Text Transfer Protocol: The protocol used by the web to transfer data.
IDE	Integrated Development Environment: Software for writing and debugging code.
IP	Internet Protocol: A set of rules for addressing and routing data.
MVC	Model View Controller: A design pattern for implementing user interfaces.
PMS	Prison Management System: Software to manage prisoner records, staff duties, and visitor appointments.
RBAC	Role-Based Access Control: Access control method based on user roles.
SRS	Software Requirements Specification: Document describing system functionalities and constraints.
SDS	Software Design Specification: A detailed design document of the software architecture.
UI	User Interface: The space where interactions between humans and machines occur.
UX	User Experience: The overall experience a user has with a product.
VC	Version Control: A system for managing changes to source code.
VS Code	Visual Studio Code: A source-code editor developed by Microsoft.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Introduction

The Prison Management System project summary is presented in this chapter. It clarifies the goals of the project as well as of the system itself, as how this chapter helps to meet the project goals. It draws attention to the need of creating an automatic and effective prison management system that concentrates on staff operations in correctional institutions, health management, visitor scheduling, and inmate records upgrade.

1.1 Problem Statement

In Pakistan and many other developing countries, correctional facilities face major challenges in managing inmate information, visitor interactions, healthcare needs, and staff operations. Traditional practices often rely on outdated manual methods like pen and paper, leading to operational inefficiencies, loss of information, security vulnerabilities, and data redundancy.
Many prisons struggle to efficiently manage visitor requests, causing long wait times and potential security issues. Similarly, tracking inmate healthcare needs becomes difficult without a centralized system, risking the health and well-being of inmates.With the growing prison population, these inefficiencies have become more critical. Studies show that facilities with poor management systems can experience operational delays up to 30%, negatively affecting inmate rehabilitation and safety outcomes .Existing systems often lack comprehensive solutions and are limited to specific functionalities like inmate tracking or communication. Therefore, a modern, integrated Prison Management System is needed to improve efficiency, inmate care, staff management, and visitor handling.This project addresses the pressing need for a scalable, secure, and centralized system to optimize prison facility workflows and promote a safer, more rehabilitative environment.

1.2 Problem Solution

The proposed Prison Management System (PMS) aims to create a comprehensive, centralized platform that integrates prisoner management, visitor management, healthcare tracking, and staff operations.
It will automate major administrative functions such as prisoner registration, appointment scheduling, complaint management, visitor request handling, and staff scheduling.
The system will use a modular architecture based on the Model-View-Controller (MVC) pattern, ensuring scalability, flexibility, and maintainability.
Objectives include improving operational efficiency, enhancing prisoner healthcare, optimizing visitor management, and ensuring role-based secure access to sensitive information.
Key goals include:

- Centralized management of visitor and prisoner files.
- Efficient tracking and scheduling of medical appointments.

- Better staff tasks distribution and supervision.
- Protect data handling with RBAC (Role-Based Access Control) and encryption.
- For jail staff members, live analytics and reporting.
- 99.9 percent uptime and strong backup/recovery policies gives systems accessibility.
- Designs of user-friendly interfaces for those without technical knowledge.

PMS allows correctional institutions much improved operational performance, guaranteed inmate welfare, and more organized and safe prison settings..

1.3 Objectives of the Proposed System

Designed to reach these particular, measurable, realistic, relevant, and time-bound (SMART) goals:

- To centralize a database for maintaining staff, inmate, and visitor records.
- To facilitate the automatic organization and monitoring of medical consultations and visits.
- To provide controlled access for various user groups running securely.
- To minimize manual paperwork and the errors it brings about by 90%
- In order to help prisoners get their grievances filed and handled effectively.
- Real-time monitoring of inmate behavior, health data, and disciplinary measures will be allowed.
- For management decision-making, precise documents offer.
- Process guest authorization requests and status adjustments in 2-3 seconds.
- Reducing disturbances maintains system uptime at 99.9 percent.
- To provide natural user interfaces that comply with accessibility rules (WCAG 2. 1).
- To preserve data accuracy via backup and recovery choices.
- To help volumes, allowing for future growth across several venues.
- To achieve system rollout and early training within the project schedule.

These objectives address reliability of the system, security of the system, operational efficiency and customer satisfaction.

1.4 Scope

The scope of the proposed Prison Management System is limited to the automation of the management of prisoner records, healthcare record tracking, visitor scheduling, staff duty scheduling and complaint handling within one correctional facility.

The key deliverable include:

- Prisoner management module (registration, record updates, cell allocation, complaints)
- Visitor management module (registration of visitors, visit requests, monitoring visits approving).
- Staff management module (staff record management, duty scheduling).
- Healthcare management module (medical requests, scheduling medical checkups).
- User authentication and role-based access control.

System Boundaries:

- Does not integrate with external government systems or databases.
- Maintained internal operations - not outgoing parole management, court administration, etc.
- Only authorized users can access the system (Administrators, Prison Staff, Prisoners, Visitors).

Stakeholders:

- Prison administrators
- Prison staff
- Inmates (Prisoners)
- Visitors (Family and Friends)
- IT Support personnel

Constraints:

- Deployment on Windows Server 2022.
- Time and budget constraints and resources will limit the initial deployment to one of two facilities.
- No AI-based predictive analytics in the first release.

Priorities:

- First: Inmate record management and security.
- Second: Visitor scheduling and approvals.
- Third: Healthcare tracking and complaint management.

Milestones:

- Requirements gathering: Month 1
- System design: Month 2
- Development and unit testing: Month 3-4
- System integration and validation: Month 5
- Deployment and user training: Month 6

1.5 System Components

The system components/modules of the proposed PMS are divided as follows

1.5.1 Module 1:Client Web App Modules:

Prisoner Dashboard

- View personal records
- Submit visit requests
- File complaints
- Request medical checkups
- Request work assignments

Visitor Dashboard

- Submit visit requests
- Check visit request status

1.5.2 Module 2: Admin Web App Modules:

Administrator Dashboard

- Register/manage prisoner records
- Allocate prisoner cells
- Schedule staff duties
- Approve/deny visitor and medical requests
- Manage complaints

Staff Dashboard

- View assigned duties
- Assign work to prisoners
- Manage medical request approvals

1.5.3 Module 3: Authentication & Security:

- User login/logout
- Password management
- Role-based access control
- Account lockout policies

1.6 Related System Analysis/Literature Review

Table 1 Related System Analysis with proposed project solution

Application Name	Weakness	Proposed Project Solution
Securus Technologies	Offers primarily inmate communications, no health care and no complete prisoner management platform	PMS offers integrated health care, staff scheduling and visitor management
Keefe Group Commissary System	Vulnerable role only in managing inmate commissary - not all aspect of facility 's operation.	PMS covers all aspects of inmate 's file, staff and health & safety.
Uganda Prison Records System	Website only record system & no visitor scheduling & complaints reporting	PMS provides extensive visitor request management & prisoner complaints
Other Prison ERP Systems	Other Prison ERP Systems often not flexible, not modular, low scalability.	PMS is having modular MVC with real time database synchronization.

1.7 Vision Statement

For prison administration officers and personnel
Who need a new, secure, and efficient method of processing inmate records, medical treatment, staff activity, and visitor interaction
The Prison Management System (PMS)
Is an end-to-end facility workflow and inmate record management solution
That automates mission-critical processes, provides secure access to sensitive information, improves operating efficiency, and facilitates the health and well-being of inmates
Unlike traditional manual systems or existing stand-alone prison ERP systems
Our product offers an integrated, extendable, and simple-to-use platform that brings inmate, staff, healthcare, and visitor activity onto one system through the use of the most current web technology.

1.8 System Limitations and Constraints

- Reliance on stable internet connection for full web-based functionality.
- Limited interoperation with external government databases due to legal and regulatory limitations.
- Limitations in compatibility on very outdated browsers or equipment for some advanced features.
- Prison regulation updating or legal compliance (e.g., GDPR, HIPAA) may require subsequent system updates.

Restrictions (Self-imposed Challenges):

- One prison building to be covered under initial roll-out only.
- No predictive analytics AI employed under initial roll-out.
- Funds and time only facilitate minimum functionality (prisoner management, visitor management, healthcare management, staff scheduling) as high-priority features.

1.9 Tools and Technologies

Table 2 Tools and Technologies for Proposed Project

	Tools	Version	Rationale
Tools And Technologies	Figma	2024	Used for prototyping, wireframing, and collaborative UI design to ensure smooth front-end development.
	Visual Studio Code	1.94	Lightweight, highly customizable editor with extensive extensions for web development and debugging.
	MS Word	2021	For writing, organizing, and formatting all project documents including reports, proposals, and manuals.
	Draw.io	20.8.16	Used for creating UML diagrams like use case diagrams, ERDs, system architecture diagrams.
	Visual Paradigm	17.2	Used for detailed UML modeling: class diagrams, sequence diagrams, component diagrams.
	MongoDB	7.0	NoSQL database system for securely storing prisoner, staff, visitor records, and complaints data in a flexible, document-oriented format.
	Technology	**Version**	**Rationale**
	Frontend Framework	React JS 18.3	Component-based library for building dynamic and responsive user interfaces efficiently.
	Backend Framework	Node.js 20.x	JavaScript runtime built on Chrome's V8 engine, enabling fast and scalable server-side development using event-driven, non-blocking I/O for building backend applications.
	API Layer	Express.js	Provides robust and flexible tools for building Web APIs in Node.js, including authentication, authorization, and data serialization.
	CSS Framework	Bootstrap 5.3	Enables responsive, mobile-first front-end development with pre-built, customizable design components.

1.10 Project Deliverables

- Project Proposal Document
- Software Requirements Specification (SRS)
- Software Design Specification (SDS)
- High-Fidelity UI/UX Prototypes
- Developed Prison Management System Application
- Final Project Report (FYP Report)

1.11 Project Planning

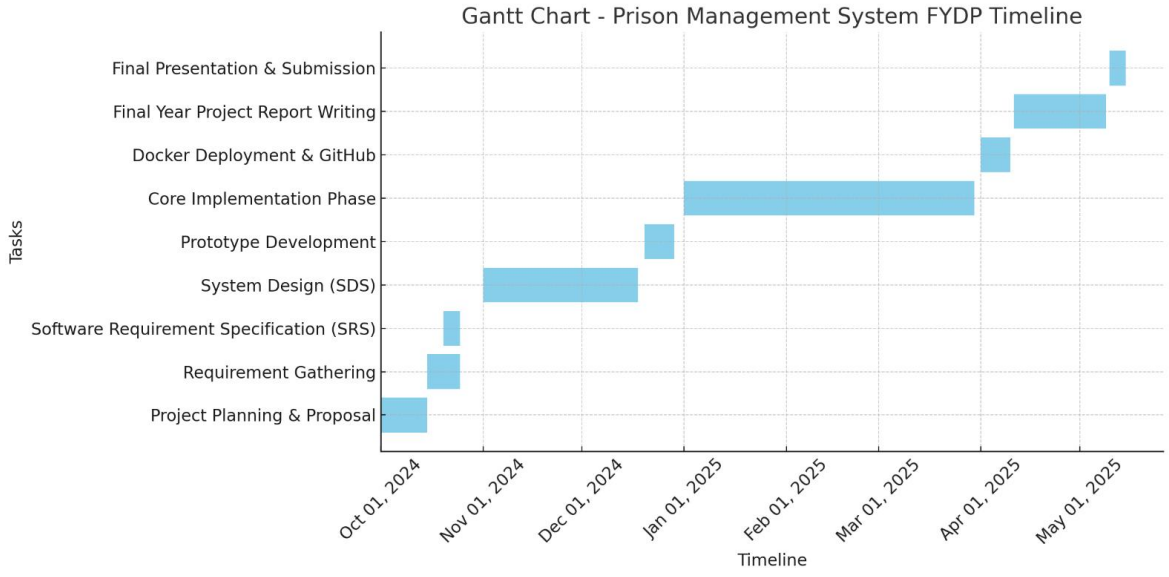


Figure 1 Gantt Chart for PMS

1.12 Summary

Chapter 1 of the PMS project sets forth the prime foundation for exposing the overview of the project about its goals, motivations, and scopes. This chapter first clarifies the core issue. Outdated manual processes in prisons in Pakistan and other similar countries add up to redundant data and inefficiency in prison processes while exposing critical security threats. The chapter further describes how manual record-keeping and lack of integration between key operations-such as inmate data, visitor scheduling, medical tracking, and staff management-leads to mismanagement causing delays for both the prison officers and the well-being of the inmates.

Centralized, secure, and modular web-based system developed-from latest technologies such as Django, React, and the storage engine MySQL-is PMS. PMS aims at automating and streamlining the key administrative functions such as: Registering-prisoners, handling complaints, appointment management and staff scheduling. It is built on SMART objectives which are meant to deliver 99.9% uptime, RBAC-based security, reduced human error, real-time monitoring, and support for user-friendly interfaces even for non-technical users.

The scope of the system is defined in Chapter 1. Apart from that, it defines internal operations to be held within a correctional facility and demarcates the boundaries for excluding other integrations externally such as court databases or parole systems. The major modules include those of prisoner, visitor, staff, healthcare, and authentication/security. Stakeholder groups are defined; and so, the system constraints such as deployment environment and time/resource limits are discussed as well.

The literature review brings out an analysis of other existing solutions and their shortcomings: non-modularity, incomplete inmate management or even none, and very few focus on handling visitors. PMS, by way of being integrated and scalable, is itself not an exclusion. The chapter ends with all-inclusive deliverables, tools and technologies that are used, and a detailed project plan on each of its development phases from requirement gathering to final deployment and training.

Chapter 2

Requirements Analysis

2. Analysis

This chapter outlines the requirements gathering and specification process for the Prison Management System (PMS). It defines the user classes, the methods used to identify requirements, and provides detailed functional and non-functional requirements for the system. Additionally, it outlines the system's external interface needs to ensure seamless interaction between users and technology components. Capturing these requirements is essential to building a solution that meets operational, security, and usability goals within correctional facilities.

2.1 User classes and characteristics

User Class	User Characteristics
Prisoner	Inmates who want to access their records, submit visitor request, request work assignment and medical checkups. They may have limited tech skills and require a user-friendly interface.
Visitor	Friends or family members of prisoner who want to request visits and check their request status.
Prison Administrator	Responsible for prison management, register prisoner and staff, remove prisoner and staff, handle and view prisoner records , visitor records and staff records, allocate cells ,and manage visitor requests.
Prison Officer	Prisoner officer is responsible for prisoner supervision, view duties schedule, view work assignments, and track the behavior of inmates and file complaints.

2.2 Requirement Identifying Technique

This section describes the requirements identifying technique(s) which further help to derive functional requirements specification. The selection of the technique(s) will depend on the type of project. For instance,

- **Use case** includes **detailed use case descriptions + use case diagram)** is an effective technique for interactive end-user applications. Use case name and associated requirements must be presented here. However, use case descriptions must be given in fully dressed format.
- **Storyboarding** for graphically intensive applications. Visual representation of sequence of events, designs to represent the flow.

2.3 Functional Requirements

User authentication:

- **FR-1:** The Administrator**,** Prison Staff**,** Prisoner**,** and Visitor shall be capable to log into the system using their unique login ID and password.

Prisoner Management:

- **FR-2:** The Administrator shall be able to register a new prisoner by entering required details such as name, date of birth, and crime details.
- **FR-3:** The Administrator shall be capable to view the details of a registered prisoner by entering their unique prisoner ID.
- **FR-4:** The Administrator shall be capable to update the details of a prisoner using their unique prisoner ID.
- **FR-5:** The Administrator shall be capable to remove a prisoner by entering their unique prisoner ID.
- **FR-6:** The Administrator shall be capable to assign a prisoner to a specific cell.
- **FR-8:** The Administrator shall be able to review complaints submitted by prisoners. **FR-10:** The Administrator shall be capable to manage visit requests from prisoners and will accept or deny it.
- **FR-14:** The Prison Staff shall be capable to assign work tasks to prisoners from a list of available tasks.
- **FR-16:** The Prisoner shall be capable to submit a request for a visit.
- **FR-17:** The Prisoner shall be capable to request specific work assignment.
- **FR-19:** The Prisoner shall be capable to file a complaint regarding any issue faced within the prison**.**

Staff management:

- **FR-7:** The Administrator shall be capable to schedule work shifts for staff members and save the information in the system.
- **FR-13:** The Prison Staff shall be capable to view their scheduled work shifts.

Visitor Management:

- **FR-9:** The Administrator shall be capable to manage visit requests made by prisoners and visitors.
- **FR-11:** Administrator shall be capable to manage visit requests from visitors and will accept or deny it.
- **FR-20:** The visitor shall be capable to submit a visit request for a particular prisoner.
- **FR-21:** The visitor shall be capable to view their visit request status**.**

Health and Medical Management:

- **FR-12:** The Administrator shall be capable to manage medical requests made by prisoners and schedule appointments as needed.
- **FR-15:** The Prison Staff shall be capable to accept or deny the requests made by the prisoners.
- **FR-18:** The Prisoner shall be capable to submit a request for medical checkup*.*

2.3.1 Functional Requirement 1 : Login

Table 3 Function Requirements Specification for login

Identifier	FR-1
Title	Login
Requirement	The Administrator, Prison Staff, Prisoner, and Visitor shall be capable to log into the system using their unique login ID and password.
Source	UC1
Rationale	To secure access to the system and assure that only protected users can log in.
Business Rule (if required)	Account will be locked after three unsuccessful login attempts.
Dependencies	None
Priority	High

2.3.2 Functional Requirement 2 : Register Prisoner

Table 4 Function Requirements Specification for Register Prisoner

Identifier	FR-2
Title	Register Prisoner
Requirement	The Administrator shall be able to register a new prisoner by entering required details such as name, date of birth, and crime details.
Source	UC2
Rationale	To maintain an up-to-date database of prisoners.
Business Rule (if required)	Duplicate entries for prisoners are not allowed.
Dependencies	FR-1
Priority	High

2.3.3 Functional Requirement 3 : View Prisoner Record

Table 5 Function Requirements Specification for View Prisoner Record

Identifier	FR-3
Title	View Prisoner Record
Requirement	The Administrator shall be capable to view the details of a registered prisoner by entering their unique prisoner ID.
Source	UC3
Rationale	To allow administrators to access prisoner information for management purposes.
Business Rule (if required)	Only authorized personnel may view prisoner records.
Dependencies	FR-1
Priority	Medium

2.3.4 Functional Requirement 4 : Update Prisoner Record

Table 6 Function Requirements Specification for Update Prisoner Record

Identifier	FR-4
Title	Update Prisoner Record
Requirement	The Administrator shall be capable to update the details of a prisoner using their unique prisoner ID.
Source	UC4
Rationale	To assure that prisoner records are current and accurate
Business Rule (if required)	Only authorized personnel may update prisoner records.
Dependencies	FR-1
Priority	Medium

2.3.5 Functional Requirement 5 : Remove Prisoner Record

Table 7 Function Requirements Specification for Remove Prisoner Record

Identifier	FR-5
Title	Remove Prisoner Record
Requirement	The Administrator shall be capable to remove a prisoner by entering their unique prisoner ID.
Source	UC5
Rationale	To maintain a clean database and remove records of released or expired prisoners.
Business Rule (if required)	Only authorized personnel van remove prisoner records.
Dependencies	FR-1
Priority	Medium

2.3.6 Functional Requirement 6 : Allocate Cells

Table 8 Function Requirements Specification for Allocate Cells

Identifier	FR-6
Title	Allocate Cells
Requirement	The Administrator shall be capable to assign a prisoner to a specific cell.
Source	UC6
Rationale	To ensure proper management of prisoner facility.
Business Rule (if required)	A prisoner can only be allocated to a cell if the cell is available.
Dependencies	FR-1
Priority	high

2.3.7 Functional Requirement 7 : Schedule Staff Duties

Table 9 Function Requirements Specification for Schedule Staff Duties

Identifier	FR-7
Title	Schedule Staff Duties
Requirement	The Administrator shall be capable to schedule work shifts for staff members and save the information in the system.
Source	UC7
Rationale	To manage staff duties effective and efficient.
Business Rule (if required)	Only authorized administrators can schedule staff shifts.
Dependencies	FR-1
Priority	Medium

2.3.8 Functional Requirement 8 : View Complaints

Table 10 Function Requirements Specification for View Complaints

Identifier	FR-8
Title	View Complaints
Requirement	The Administrator shall be able to review complaints submitted by prisoners.
Source	UC8
Rationale	To address prisoner complaints efficiently.
Business Rule (if required)	Only authorized administrators can view the complaints.
Dependencies	FR-1
Priority	Medium

2.3.9 Functional Requirement 9 : Manage Visit Request

Table 11 Function Requirements Specification for Manage Visit Request

Identifier	FR-9
Title	Manage Visit Request
Requirement	The Administrator shall be capable to manage visit requests made by prisoners and visitors.
Source	UC9
Rationale	To regulate visiting rights and to maintain and regulate security within the system.
Business Rule (if required)	Visit requests must comply with institutional policies.
Dependencies	FR-1
Priority	High

2.3.10 Functional Requirement 10 : Manage Prisoner Visit Request

Table 12 Function Requirements Specification for Manage Prisoner Visit Request

Identifier	FR-10
Title	Manage Prisoner Visit Request
Requirement	The Administrator shall be capable to manage visit requests from prisoners and will accept or deny it.
Source	UC10
Rationale	To manage requests received by the prisoners.
Business Rule (if required)	Approved requests must be communicated to both the prisoner and the visitor.
Dependencies	FR-1 , FR-9
Priority	High

2.3.11 Functional Requirement 11 : Manage Visitor Visit Request

Table 13 Function Requirements Specification for Manage Visitor Visit Request

Identifier	FR-11
Title	Manage Visitor Visit Request
Requirement	The Administrator shall be capable to manage visit requests from prisoners and will accept or deny it.
Source	UC11
Rationale	To manage requests received by the visitors.
Business Rule (if required)	Approved requests must be communicated to both the prisoner and the visitor.
Dependencies	FR-1 , FR-9
Priority	High

2.3.12 Functional Requirement 12 : Manage Medical Request

Table 14 Function Requirements Specification for Manage Medical Request

Identifier	FR-12
Title	Manage Medical Request
Requirement	The Administrator shall be capable to manage medical requests made by prisoners and schedule appointments as needed.
Source	UC12
Rationale	To provide necessary healthcare services to prisoners in a timely manner.
Business Rule (if required)	All medical requests must comply with healthcare regulations.
Dependencies	FR-1
Priority	high

2.3.13 Functional Requirement 13 : View Duties Schedule

Table 15 Function Requirements Specification for View Duties Schedule

Identifier	FR-13
Title	View Duties Schedule
Requirement	The Prison Staff shall be capable to view their scheduled work shifts.
Source	UC13
Rationale	To keep staff informed of their assigned duties.
Business Rule (if required)	Staff members can only view their own scheduled duties.
Dependencies	FR-1
Priority	Medium

2.3.14 Functional Requirement 14 : Assign Work

Table 16 Function Requirements Specification for Assign Work

Identifier	FR-14
Title	Assign Work to prisoners
Requirement	The Prison Staff shall be capable to assign work tasks to prisoners from a list of available tasks.
Source	UC14
Rationale	To provide productive activities for prisoners.
Business Rule (if required)	None
Dependencies	FR-1
Priority	Medium

2.3.15 Functional Requirement 15 : Permit Medical Request

Table 17 Function Requirements Specification for Permit Medical Request

Identifier	FR-15
Title	Permit Medical Request
Requirement	The Prison Staff shall be capable to accept or deny the requests made by the prisoners.
Source	UC15
Rationale	To provide necessary healthcare services to prisoners in a timely manner.
Business Rule (if required)	Only staff members can accept or deny it.
Dependencies	FR-1
Priority	high

2.3.16 Functional Requirement 16 : Request Visitor

Table 18 Function Requirements Specification for Request Visitor

Identifier	FR-16
Title	Request Visitor
Requirement	The Prisoner shall be capable to submit a request for a visit.
Source	UC16
Rationale	To facilitate communication and visits between prisoners and their approved visitors.
Business Rule (if required)	Requests must comply with visitation policies.
Dependencies	FR-1
Priority	high

2.3.17 Functional Requirement 17 : Request Work Assignment

Table 19 Function Requirements Specification for Request Work Assignment

Identifier	FR-17
Title	Request work Assignment
Requirement	The Prisoner shall be capable to request specific work assignment.
Source	UC17
Rationale	To engage prisoners in meaningful work.
Business Rule (if required)	Prisoners can only request work assignments that are available.
Dependencies	FR-1
Priority	Medium

2.3.18 Functional Requirement 18 : Request Medical Checkup

Table 20 Function Requirements Specification for Request Medical Checkup

Identifier	FR-18
Title	Request Medical Checkup
Requirement	The Prisoner shall be capable to submit a request for medical checkup.
Source	UC18
Rationale	To ensure that prisoners have access to medical care when necessary..
Business Rule (if required)	Medical requests must be reviewed by authorized staff before approval.
Dependencies	FR-1
Priority	High

2.3.19 Functional Requirement 19 : File Complaints

Table 21 Function Requirements Specification for File Complaints

Identifier	FR-19
Title	File Complaints
Requirement	The Prisoner shall be capable to file a complaint regarding any issue faced within the prison.
Source	UC19
Rationale	To allow prisoners to voice their grievances and maintain oversight.
Business Rule (if required)	Complaints must be submitted through the official channel.
Dependencies	FR-1
Priority	Medium

2.3.20 Functional Requirement 20 : Request Visit

Table 22 Function Requirements Specification for Request Visit

Identifier	FR-20
Title	Request Visit
Requirement	The visitor shall be capable to submit a visit request for a particular prisoner.
Source	UC20
Rationale	To manage visitation efficiently while keeping records of requests.
Business Rule (if required)	Visitors must be registered in the system before submitting a visit request.
Dependencies	FR-1
Priority	High

2.3.21 Functional Requirement 21 : View Request Status

Table 23 Function Requirements Specification for View Request Status

Identifier	FR-21
Title	View Request Status
Requirement	The visitor shall be capable to view their visit request status.
Source	UC21
Rationale	To manage visitation efficiently.
Business Rule (if required)	Visitors can only view their status of request.
Dependencies	FR-1
Priority	High

2.4 Non-Functional Requirements

The non-functional requirements define system quality attributes.

2.4.1 Reliability

- Mean Time between Failures (MTBF) should exceed 800 hours.
- Mean Time to Recover (MTTR) from any system failure should be less than 1 hour.
- Backup policies include full daily backups and hourly incremental backups.
- Error logging and admin notifications must happen within 5 minutes of failure detection.

2.4.2 Usability

- Administrators must be able to complete visit scheduling or prisoner registration within 3–4 interactions.
- The system must comply with WCAG 2.1 Accessibility Standards for visually impaired users.
- First-time users should become proficient within 3 hours of hands-on practice with the system.
- Clear, helpful error and confirmation messages should be provided.

2.4.3 Performance

- Implement Role-Based Access Control (RBAC) to restrict system access by user type.
- Encrypt all data transmissions using TLS 1.3.
- Encrypt sensitive database fields (like prisoner health records) with AES-256.
- Audit logs must capture all critical user activities and be retained for at least 1 year.
- System must enforce account lockout after 3 consecutive failed login attempts.

2.4.4 Security

- Implement Role-Based Access Control (RBAC) to restrict system access by user type.
- Encrypt all data transmissions using TLS 1.3.
- Encrypt sensitive database fields (like prisoner health records) with AES-256.
- Audit logs must capture all critical user activities and be retained for at least 1 year.
- System must enforce account lockout after 3 consecutive failed login att

2.5 External Interface Requirements

2.5.1 User Interfaces Requirements

Material design by Google outlines specific design rules, such as those for layout, navigation, components, and usability, to help create standardized and intuitive interfaces .Despite being frequently used for online and mobile applications, its straightforward form makes it a useful guide for system design.

1. Standards for Fonts, Icons, Button Labels, Color Scheme and Controls:

- **Fonts:** Use clear sans-serif fonts such as Arial, Open Sans to ensure readability.
- **Icons**: Standard icons will be used for actions such as save, delete, and search, ensuring they are intuitive and recognizable.
- **Button labels**: Button labels for common actions such as Add Prisoner, View Record administrative actions such as login, logout and general navigation such as home, back are used.
- **Color Scheme:** Use Black or Blue colors for headers and important UI elements and white as background color. Red for warnings and alerts.
- **Controls:** Commonly used controls like radio buttons for gender selection (male/female), date picker for DOB selection, and tool tips are included.

2. Screen layout and resolution restrictions:

The responsive design works with a range of devices, including tablets and desktop computers. A header, navigation bars, and footers are used on every screen to arrange the content. Increase the resolution of your design to improve visibility and clarity.

3. Standard buttons and navigation links:

- **Primary Action Buttons:** Used for main actions like submit etc.
- **Secondary action Buttons:** Used for actions that are less important like Back etc.
- **Navigation Links:** Links to the primary sections like Home, Settings and Help are included.

4. Keyboard shortcuts:

Shortcut keys used in Prison Management system:

- **Ctrl I**: Creates a new item.
- **Ctrl S:** Save current changes.
- **Ctrl H**: Quickly navigate back to home page.
- **Ctrl C**: Copy selected information.

5. Message display conventions:

- **Error messages:** Clear and meaningful error messages of our system are provided such as Prisoner ID not found. Please check and try again.
- **Confirmation Messages:** These messages clearly tells the user that action was completed successfully like Record updated successfully.
- **Message Formatting:** Colors like red is used for error messages, yellow for warning messages and green for success messages.

6. Layout standards for localization:

The interface supports language (English) with a flexible layout that supports text expansion and right-to-left reading as needed.

7. Adaptation for visually impaired users:

All UI elements are labeled for screen readers. Full keyboard navigation is supported.

2.5.2 Software interfaces

1. Applications:

- **VS Code (Version 1.94):**
  Required for creating and managing the prison management system's front-end and back-end .Coders collaborate with the system by VS Code for writing, debugging, and testing of the code.
- **Figma (Version 2024):**
  It is a UI/UX tool for creating storyboards and user interfaces. Designs made in Figma are merged into the front-end using React.js. Coders import design into VS Code for UI implementation.
- **MS Word (Version 2021):**
  It is used for documentation of the project. It provides an environment for developing and exchanging project plans and documentations between developers and stakeholders.

2. Database:

- **MySQL (Version 8.0.39):**
  It act as the primary database for storing Prisoners records, visitor calls, health data, and Prison staff information. It interacts with the Django for database queries, using built-in database management capabilities.

3. Tools:

- **Draw.io (Version 20.8.16):**

  It is a diagramming tool for representing framework and use cases.Diagrams are merged into documentation and used by developers during the system design.

- **Visual Paradigm (Version 17.2):**

  It is a another diagramming tool for representing framework and use cases. Diagrams are merged into documentation and used by developers during the system design.

4. Libraries :

- **Django REST Framework (Version 4.0):**

  This framework is used to show Application Programming Interface for connection between the front-end and back-end .Provides flattening , validation, and user management. Front-end requests are sent through APIs revealed by Django on Windows OS.

- **React Router (Version 6.0):**

  Allow vibrant routing between different pages in the system e.g Prisoner staff dashboards, Prisoner profiles, and visitor management views .React Router manages browsing between these modules, assuring fluid transformation and a fluid customer interaction.

- **Bootstrap 5.3:**

  Front-end CSS framework used for styling and responsive design. Its pre-built classes are used to create user-friendly and responsive interfaces.

- **React.js (Version 18.3):**

  It is used in making front-end of the system that communicates with the back-end through API calls. Its components are designed in VS Code and deployed on Internet Information Services. Front-end communication with Django are managed with Django REST Framework.

5. Operating System

- **Windows Server 2022:**

  All the tools and libraries used in the system are implemented and supervised within the Windows Server atmosphere .The Windows Operating system handles execution of the processes, task and resource management and security of the system.

2.5.3 Hardware interfaces

1. Server Hardware

Supported Device Types: Window server.

- The Prison Management System backend which will be built by using Django will be operated on specialized server. The server will manage all arriving demands from the users, and it will handle data, communicate with the database, and give the related output.
- The hardware of the server should back database and instant communications.

Communication Protocols:

- The server of our Prison Management System will interact with client devices including a PCs mobile devices by HTTP/HTTPS protocols.
- Protected network will be guaranteed using Transport Layer Security to encode communication between the server and client devices.

2. Client Devices

Supported Device Types: Windows-based PCs, tablets, and mobile phones.

- Users including Prison Administrators, Staff, Visitors, prisoners will use the system by using web browsers on different devices. The web interface should be dynamic and consistent with many screen sizes.
- Devices used by the Client will send data to the server including visitor requests, Prisoner requesting for work assignments and will obtain data or approvals through a web interface.

Communication Protocols:

- Client and server use HTTP/HTTPS protocols for protected communication.
- Data from client will be sent to the server by using Restful API calls, assuring protected and effective data transmission.

2.5.4 Communications interfaces

1. Email Communication:

The system will send email notifications to the users of the system including Prison administrators, Prison staff, visitors, and prisoners automatically when any of the events such as visitor permissions, checkup and system alerts.

Requirements:

- SMTP (Simple Mail Transfer Protocol)
- HTML Email Templates
- TLS/SSL

2. Web Browser :

The system will provide a fluid web-based user interface available through modern web browsers.

Requirements:

- HTTP/HTTPS
- Bootstrap
- React Js

3.Network Protocols:

The system will communicate across protected network protocols to protect confidential and sensitive data during transmission.

Requirements:

- HTTPP/HTTPS
- TCP/IP
- Web-sockets

4.Electronic Forms:

The system will use electronic forms for data entry and different management of the tasks including visitor request forms, health checkup requests.

Requirements:

- React Js
- JavaScript

2.6 Summary

Chapter 2 acts as the building block of the Prison Management System (PMS), giving a thorough exposition of the system's functional and non-functional requirements in consonance with the objectives of the introduction to the project. The chapter begins with an extensive description of the user roles in the system, namely administrator, prison staff, prisoner, and visitor as well as their operational needs translated into well-defined system behaviors through a clear set of use cases. Each use case stands for a significant feature or process that the system must support for prisoner registration, scheduling staff duty, processing medical requests, and scheduling visitor entries and approval and managing complaints submitted by inmates. These use cases are designed not only in such a way as to make evident the interactive flows of the system but with all requisite items like actors involved, preconditions, main and alternate flows, postconditions, and exception handling, thus clarified for the developers during implementation.

To go beyond what the system is intended to do, the chapter describes how the system should behave and perform by capturing non-functional requirements. Among these include strict security measures such as role-based access control (RBAC), encrypted communications, user authentication protocols, and audit trails for monitoring user activity-all basics because of the sensitivity brought by prison data. The other performance goals are minimal system latency, 99.9% uptime, fast request processing (especially for visitor approvals and inmate queries), and scalable architecture in place, for the system would still have to become resilient and responsive in usage increase or expansion to other facilities. Usability and accessibility have also been a topic of interest, with an assurance that a user-friendly interface adaptable for use by non-technical staff will be constructed and compliant with the web accessibility standards (WCAG 2.1). Other aspects like system reliability, maintainability, and interoperability limitations are well covered so that realistic boundaries and expectations for development and deployment are firmly established.

Overall, this chapter authorizes into real and structured needs the vision of the Prison Management System vis-in-vis making sure anything down from high-level workflows to technical constraints gets documented and justified towards developing such a user-friendly, secure, and effective digital solution in managing contemporary correctional facilities.

Chapter 3

Design and Architecture

3. System Design

3.1 Design considerations

Assumptions and Dependencies:

- **User Familiarity with Technology:** The system assumes basic computer literacy and familiarity with the web-based applications to be possessed by the majority of the prison staff and administrations. The interface in this case is therefore intended to be user-friendly and simple.
- **Internet Availability:** The system has been designed to allow access through web browsers notwithstanding the fact that internet access is required for viewing the system remotely by visitors and also for administrative use within the prison. In the case of unreliable internet connections, such as those that are encountered in prisons, a combination of local instillations or offline options may be prescribed.
- **Data Synchronization:** The assumption is made that data pertaining to different modules (modules like the prisoner files, the medical files, the records of work assignments, managerial files etc) will be gathered and harmonized across the different modules in real time so as to eliminate data inconsistencies.

Limitations:

- **Restrictions on Integration with Other Systems**: Limitations may also occur as to integrating with external entities such as government systems as a result of legal limitations, data-sharing agreements, and technical features of those external systems. Such restrictions may affect the ability for cross-platform updates of prisoner data in real-time.
- **Single Server Dependence in the Network:** The system does not implement a caching mechanism or offline mode for the client application. In normal circumstances, however, tasks that require updating prisoner records or processing of visit requests may be carried out over relayed services once the network is re-established.
- **User Constraints of the Interface:** Some other features may persist in being unsupported by some older versions of web browsers or by some mobile devices, thus, system design will be intended to rely on web browser residuality even though some features of the system will be incompatible with certain new versions or models of web browsers.

Risks:

1. Security Risks:

a. Data Breach: The sensitive data will contain personal information and health records of prisoners, hence becomes a target for malicious individuals. In this regard, the system will ensure that all sensitive data stored and transmitted are encrypted, MFA for all those with access to the system, and have regular security audits.

b. Insider Threats: Incarceration officers or management may misuse the facility of access. This is taken care of by including RBAC and user activity logs with minute details.

2. Scalability Risks:

High Volume of Traffic: Performance may occur during high usage like when prisoners are processed or there are many visits on a given day. The design will have load balancing and cloud infrastructure to handle high volume peaks in traffic volume and Scalability.

3. Regulatory Risks:

Legal Compliance: The data of inmates is hugely sensitive to legal compliance about practices such as GDPR and HIPAA. The system design needs to accommodate anonymization, access controls, and scheduled compliance audits that it will be able to ensure adherence to the above-mentioned requirements.

4. System Outage:

Hardware or Software Failures: Any downtime can severely cripple prison operations. As a mitigating feature, the system design includes fail-over mechanisms, regular backups, and the ability to quickly restore the system from a backup

5.User Training and Adoption:

Resistance to the New System: Some users might resist it because they have never used a system like this. Risk could be minimized through proper and more elaborate user training support and system feedback over time.

3.2 Design Models

Provide the descriptions of the following models used to describe the system design. Also ensure visibility of all diagrams.

The applicable models for the project using object-oriented development approach may include:

1. Class Diagram
2. Interaction Diagram (Either sequence or collaboration)
3. State Transition Diagram (for the projects which include event handling and backend processes)

3.3 Architectural Design

The design of the Prison Management System (PMS) architecture will be according to the Layered Architecture using MVC principles, thus the architecture of the Prison Management System will be truly be as natural one.

1. Overview of Layered Architecture

The system can be divided into four significant layers:

- **Presentation Layer:** This layer contains the user interfaces of both administrators, prison staff, prisoners, and visitors.
- **Business Logic Layer:** Manages core application functionalities. This can include prisoner registration, cell allocation, approval of medical requests, and complaints handling.
- **Database Layer:** It is the relational database where all data are therefore stored.

2. UML Component Diagram

The Component Diagram emphasizes the modular characteristic of the PMS and shows the relationship between components.

Main components:

- **User Interface Component:** Dedicated interfaces for administrators, staff, prisoners, and visitors.
- **Authentication Component:** Manages login and session management.
- **Prisoner Management Component:** Manages prisoner records, updates, cell assignments, and work assignments.
- **Staff Management Component:** Responsible for management of staff schedules and work assignment allocation.
- **Visitor Management Component:** Manages the visit requests and status.
- **Complaint Management Component:** Submits and reviews complaints.
- **Health Management Component:** Handles medical requests and schedules.
- **Data Access Component:** Acts as an abstraction layer for database operations.
- **Database:** Stores all system data.

3. UML Class Relationship Diagram:

Class Diagram provides a class diagram of the system's object oriented structures, with the following:

- **User:** A base class from which other sub-classes inherit: Administrator, Prison Staff, Prisoner, and Visitor.
- **Prisoner:** Attributes-name, prisoner ID, cell assigned, medical history, work status.
- **Visitor:** Attributes-visitor ID, associated prisoner, visit requests.
- **Complaint:** Attributes-complaint ID, prisoner ID, description, and status.

- **Medical-Request:** Attributes include request ID, prisoner ID, health issue, and approval status.
- **Schedule:** Attributes include staff schedules, prisoner work assignments, and cell allocation.

4. UML DIAGRAMS

UML Component Diagram

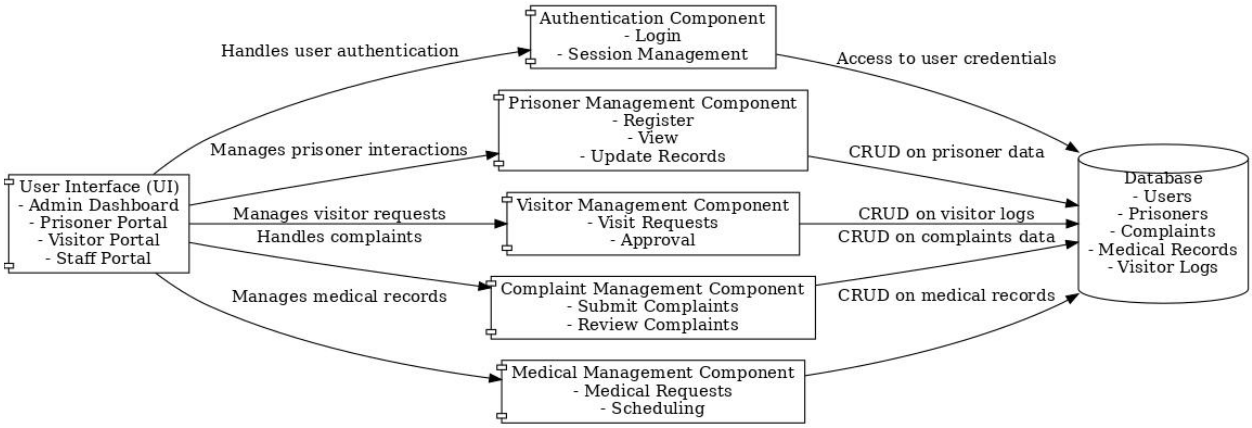


Figure 2 UML Component Diagram

UML Class Relationship Diagram

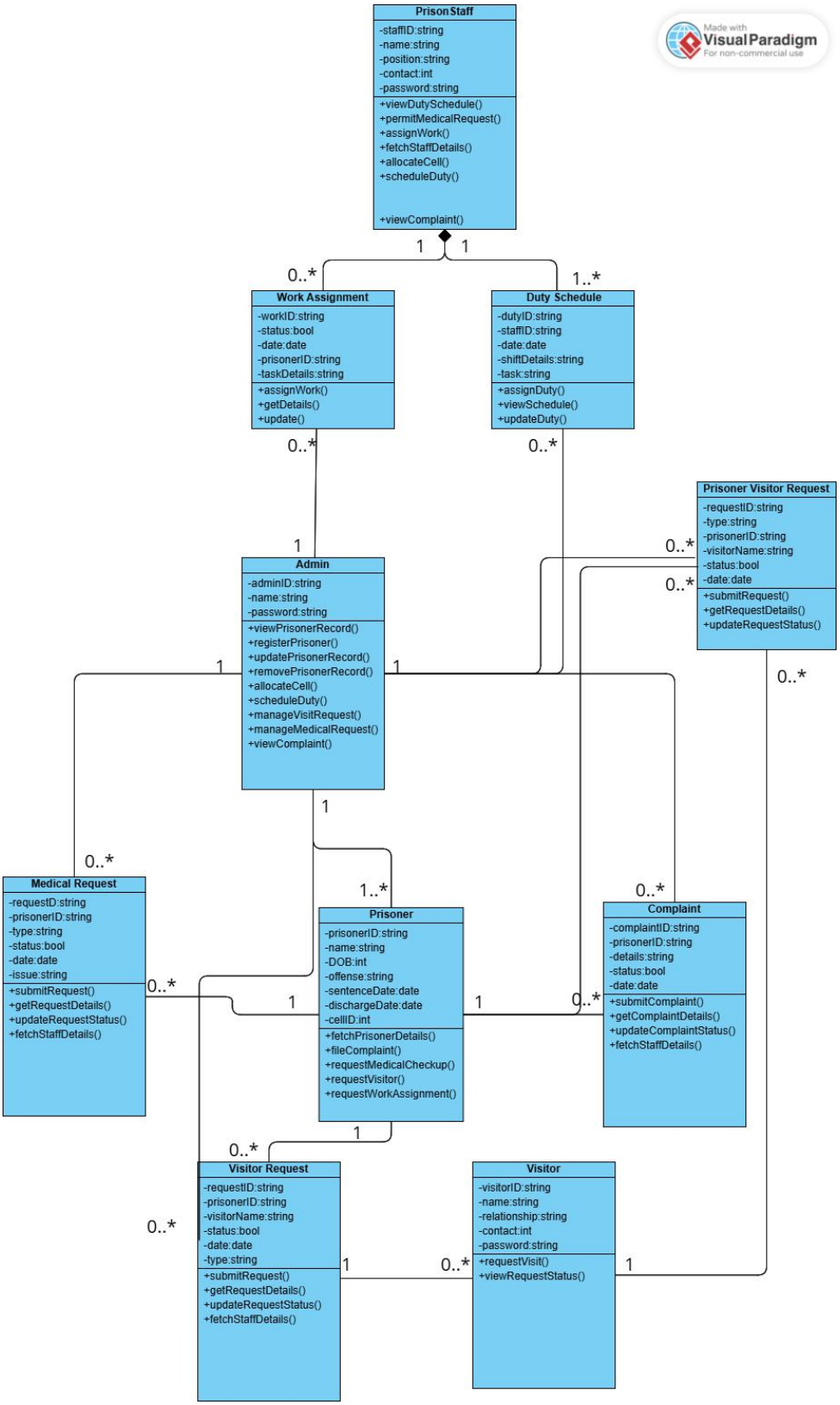


Figure 3 UML Class Relationship Diagram

5. Mapping Components to Architecture :

Table 24 Mapping components to Architecture

Layer	Modules/Components	Responsibilities	Technology
Presentation Layer (View)	• Prisoner Dashboard • Visitor Dashboard • Staff Dashboard • Admin Dashboard	• Provides user interfaces for different users (Prisoner, Visitor, Staff, Admin). • Collects inputs from users and displays processed results.	React Js
Business Logic Layer (Controller)	• Authentication • Prisoner management • Visitor management • Request Management • Complaint Management • Medical Request Management	• Handles business logic for user authentication and role-based access. • Manages requests for visits, complaints, medical checkups, and work assignments. • Controls duty scheduling for staff.	Django
Database Layer	• Prisoner Repository • Visitor Repository • Staff Repository • Medical Request Repository • Complaint Repository • Schedule Repository • Work Assignment Repository	• Interacts with the database to execute queries and retrieve data for higher layers. • Provides an abstraction of the database for the business logic layer.	MySQL

3.4 Data Design

1. Information Domain Transformation

The system's primary entities, derived from the use cases, are presented as relational tables in the database. Each entity corresponds to a class in the object-oriented model.

2. Data Flow and Processing

2.1 Data Entry

Data is entered into the system through user interfaces (UI) designed for administrators, staff, prisoners, and visitors.

2.2 Data Processing

Each module processes its related data:

- **Prisoner and Visitor Modules**: Handles prisoners and visitors requests.
- **Staff Module**: Handles staff duties.
- **Administrator Module**: Manages overall data and work-flows.

2.3 Data Retrieval

Data is fetched based on user roles:

- Prisoners fetch their own data
- Staff views duty schedules.
- Visitors track visit request statuses.
- Administrator controls all data for management purposes.

3. Data Structures

Data set/Database Design:

The database design follows the **relational database model**, normalized to **3rd Normal Form (3NF)** for efficiency and data integrity.

3.4.1 Data Dictionary

- **Alphabetical List of System Entities or Major Data with Types and Descriptions:**

Table 25 Data Dictionary

Term (Entity/Attribute/Method)	Description (Data Type, Role)
adminID	Unique identifier for each Administrator (**string**).
allocateCell(prisonerID, cellID)	Method: Assigns a cell to a prisoner (**string, string**).
assignDuty(staffID, shiftDetails)	Method: Assigns duties to staff members (**string, string**).
assignWork(assignedTo, taskDetails)	Method: Assigns work tasks to prisoners (**string, string**).
cellID	Identifier for the allocated cell of a Prisoner (**string**).
complaintID	Unique identifier for a Complaint (**string**).
contact	Contact information for Prison Staff (**string**).
date	Represents the date of an event, e.g., submission or assignment (**date**).
details	Description or content of a Complaint (**string**).
dischargeDate	Expected release date for a Prisoner (**date**).
DOB	Date of birth of a Prisoner (**date**).
dutyID	Unique identifier for a DutySchedule (**string**).
fetchPrisonerDetails(prisonerID)	Method: Retrieves details of a specific prisoner (**string**).
fetchStaffDetails(staffID)	Method: Retrieves details of a specific staff member (**string**).
fileComplaint(prisonerID)	Method: Allows a prisoner to file a complaint (**string**).
gender	Gender of a Prisoner (**string**).

Term (Entity/Attribute/Method)	Description (Data Type, Role)
getComplaintDetails(complaintID)	Method: Retrieves details of a specific complaint (**string**).
getDetails(workID)	Method: Retrieves details of a work assignment (**string**).
getRequestDetails(requestID)	Method: Retrieves details of a request (**string**).
issue	Description of a medical issue in a MedicalRequest (**string**).
manageMedicalRequest(requestID)	Method: Allows an Administrator to manage medical requests (**string**).
manageVisitRequest(requestID)	Method: Allows an Administrator to manage visitor requests (**string**).
name	Name of an entity (**string**). Used for Prisoners, Staff, Visitors, etc.
offense	Crime committed by a Prisoner (**string**).
password	Password for login authentication (**string**).
permitMedicalRequest(requestID)	Method: Allows PrisonStaff to approve medical requests (**string**).
position	Designation of a Prison Staff member (**string**).
prisonerID	Unique identifier for a Prisoner (**string**).
registerPrisoner(name, age, gender, offense, sentenceDate, dischargeDate)	Method: Allows an Administrator to register a new prisoner (**string, integer, string, string, date, date**).
relationship	Relationship of a Visitor to the Prisoner (**string**).
removePrisonerRecord(prisonerID)	Method: Removes a Prisoner's record (**string**).
requestID	Unique identifier for a request (**string**).
requestMedicalCheckup(prisonerID)	Method: Allows a Prisoner to request a medical checkup (**string**).
requestVisitor(prisonerID)	Method: Allows a Prisoner to request a visitor (**string**).
requestVisit(visitorID)	Method: Allows a Visitor to request a visit to a Prisoner (**string**).

Term (Entity/Attribute/Method)	Description (Data Type, Role)
scheduleDuty(staffID, shiftDetails)	Method: Allows an Administrator to schedule duties for staff (**string, string**).
sentenceDate	Start date of a Prisoner's sentence (**date**).
shiftDetails	Description of the assigned shift in DutySchedule (**string**).
staffID	Refers to the unique identification of a staff member (string).
start	Refers to the date or time of the timestamp for different events (date/time).
status	Represents a string for the current status of tasks, complaints, and requests.
submitComplaint(prisonerID, details)	Method: This allows a prisoner to submit a complaint (string, string).
submitRequest(prisonerID, issueDetails)	Method: Allows a prisoner to submit a medical request (**string, string**).
task	Description of a task in DutySchedule (**string**).
taskDetails	Description of work assigned to a Prisoner (**string**).
type	Type of request, e.g., medical or visitor (**string**).
update(workID)	Method: Updates a work assignment (**string**).
updateComplaintStatus(complaintID)	Method: Updates the status of a complaint (**string**).
updatePrisonerRecord(prisonerID, updatedDetails)	Method: Updates the details of a Prisoner (**string, object**).
updateRequestStatus(requestID)	Method: Updates the status of a request (**string**).
viewComplaint(complaintID)	Method: Allows an Administrator to view a complaint (**string**).
viewDutySchedule(staffID)	Method: Allows PrisonStaff to view assigned duties (**string**).
viewPrisonerRecord(prisonerID)	Method: Allows an Administrator to view prisoner details (**string**).

- **Object-Oriented(OO) Approach: Objects, Attributes, Methods & Method Parameters:**

1. Object: Prisoner

Attributes:

- **prisonerID**: Unique identifier for a prisoner.
- **password:** password for login security.
- **name**: Name of the prisoner.
- **DOB**: Date of birth of the prisoner.
- **gender**: Gender of the prisoner.
- **offense**: Crime committed.
- **sentenceDate**: Date when the sentence started.
- **dischargeDate**: Expected release date.
- **cellID**: Identifier for the allocated cell.

Methods:

- *fetchPrisonerDetails(prisonerID)*
- *fileComplaint(prisonerID)*
- *requestMedicalCheckup(prisonerID)*
- *requestVisitor(prisonerID)*
- *requestWorkAssignment(prisonerID)*

2. Object: Prison Staff

Attributes:

- **staffID:**Unique identifier for a staff
- **name:** Name of staff person
- **position:** Designation of prison officer
- **contact:** Includes phone number .
- **password:** password for login security.

Methods:

- *viewDutySchedule(staffID)*
- *permitMedicalRequest(requestID)*
- *assignWork(prisonerID,workDetails)*
- *fetchStaffDetails(staffID)*

3. Object: Administrator

Attributes:

- **adminID:** Unique identifier for each administrator.
- **name:**Name of the administrator.
- **password:** password for the administrator's account.

Methods:

- *viewPrisonerRecord(prisonerID)*
  - *RegisterPrisoner(name, age, gender, offense, sentenceDate, dischargeDate)*
  - *updatePrisonerRecord(prisonerID, updatedDetails)*
  - *removePrisonerRecord(prisonerID)*
  - *allocateCell(prisonerID, cellID)*
  - *scheduleDuty(staffID, shiftDetails)*
  - *manageVisitRequest(requestID,requestType)*
  - *manageMedicalRequest(requestID)*
  - *viewComplaint(complaintID)*

4. Object: Visitor

Attributes:

- **visitorID:** Unique identifier for each visitor.
- **name:** Name of the visitor.
- **relationship:** The visitor's relationship to the prisoner
- **password:** password for the visitor's account.
- **phone:** Visitor's phone number.

Methods:

- *requestVisit(visitorID)*
- *viewRequestStatus(requestID)*

5.Object: Complaint

Attributes:

- **complaintID:** Unique identifier for each complaint.
- **prisonerID:** The ID of the prisoner who submitted the complaint.
- **details:** The content of the complain
- **status:** Current status of the complaint
- **date:** The date the complaint was submitted.

Methods:

- *submitComplaint(prisonerID, details)*
- *getComplaintDetails(complaintID)*
- *updateComplaintStatus(complaintID)*

6. Object : MedicalRequest

Attributes:

- **requestID:** Unique identifier for medical request.
- **type:** specifies the type of request.
- **prisonerID:** The ID of the prisoner submitting the medical request.
- **issue:** A description of the medical issue faced by the prisoner.
- **status:** Current status of the medical request.
- **date:** Date the request is submitted.

Methods:

- *submitRequest(prisonerID,issueDetails)*
- *getRequestDetails(requestID)*
- *updateRequestStatus(requestID)*

7. Object: PrisonerVisitorRequest

Attributes:

- **requestID:** Unique identifier for request.
- **type:** specifies the type of request.
- **prisonerID:** The ID of the prisoner
- **visitorName:** Name of the visitor requested by the prisoner.
- **status:** Current status of the request.
- **date:** Date the request is submitted.

Methods:

- *submitRequest(prisonerID,visitor name)*
- *getRequestDetails(requestID)*
- *updateRequestStatus(requestID)*

8. Object: visitorVisitRequest

Attributes:

- **requestID:** Unique identifier for request.
- **Type:** specifies the type of request.
- **prisonerID:** Unique identifier for the prisoner.
- **visitorID:** Unique identifier for the visitor.
- **status:** Current status of the request.
- **date:** Date the request is submitted.

Methods:

- *submitRequest(prisonerID,visitorID)*
- *getRequestDetails(requestID)*
- *updateRequestStatus(requestID)*

9. Object: workAssignment

Attributes:

- **workID:** Unique identifier for the work
- **status:** Status of the work task
- **date:** Date of assignment
- **prisonerID:** identifier for the prisoner receiving the assignment.
- **taskDetails:** Description of the work task.

Methods:

- *assignWork(assignedTo,taskDetails)*
- *getDetails(workID)*
- *update(workID)*

10. Object: DutySchedule

Attributes:

- **dutyID:** Unique identifier for the duty.
- **staffID:** ID of the staff assigned to the task
- **shiftDetails:** Shift assigned for the task.
- **task:** Description of the task or duty.
- **Date:** date of assignment.

Methods:

- *assignDuty(staffID, shiftDetails)*
- *viewSchedule(staffID)*
- *updateDuty(dutyID, dutyDetails,staffID)*

3.5 User Interface Design

In "Prison Management System", it has four user roles accepting the administrator, staff, prisoners, and visitors, to represent an interface that is fairly rigorous yet almost intuitive.

Administrators:

- **Functions:** Manages prisoners' records (adding, updating, and deleting), cell allocation, staff duty scheduling, visitor scheduling, medical requests, and complaints.
- **Feedback:** Alert notification from the system has been gained for those tasks that are waiting for their further processing as well as for the system reports and request status.

Prison Staff:

- **Features:** Check and manage the staff scheduling, let prisoners work, if medically approved, and trace the complaints resolving process overall problem timeline.
- **Feedback:** Real-time updates concerning duty assignments, prisoner assigned tasks, and medical approval.

Prisoners:

- **Functions:** Raising requests for visitor meetings, medical checkups, and assignment of work. Complaints can also be registered here, and the fate of all requests may be viewed.
- **Feedback:** Notices on request acceptance, rejection, or follow-ups.

Visitors:

- **Functions:** Request a visit, check status, and access information on scheduled visits.
- **Feedback:** Email or portal notifications about approval or rescheduled statuses. The system works to ensure that all activities are done by real-time feedback in a very easy-to-navigate manner. Thus, dashboards for each role provide actionable insights through enjoyable experience.

3.3.1 Screen Images

Admin Dashboard

Figure 4 Admin Dashboard

Login Page

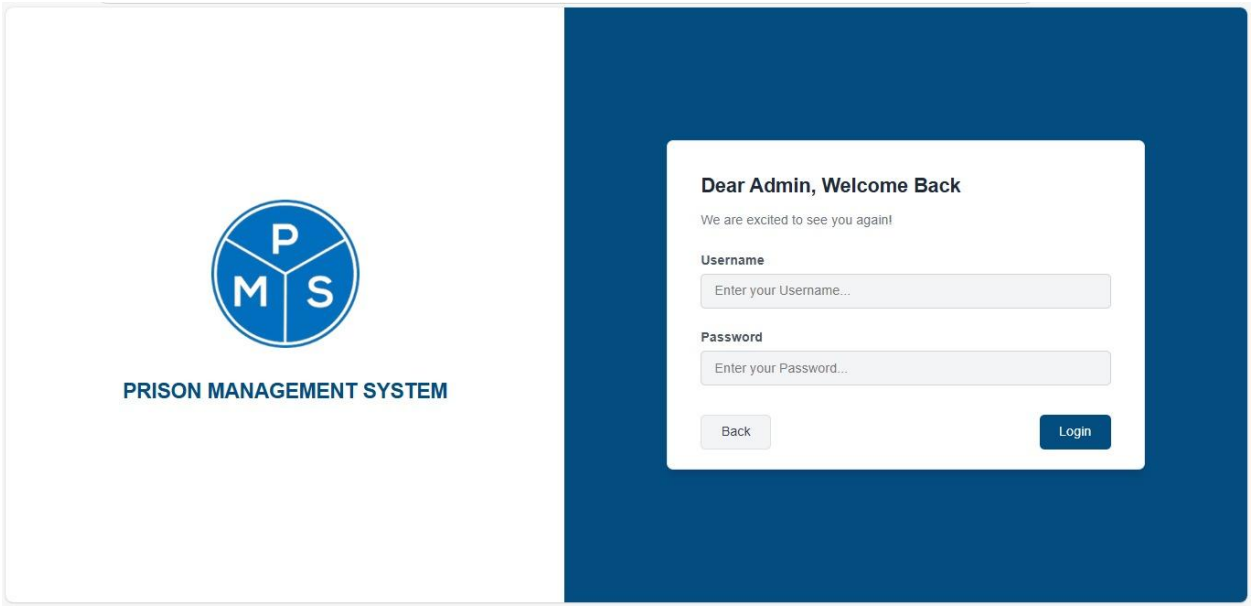


Figure 5 Login Page

Inmate Dashboard

Figure 6 InmateDashboard

Staff Dashboard

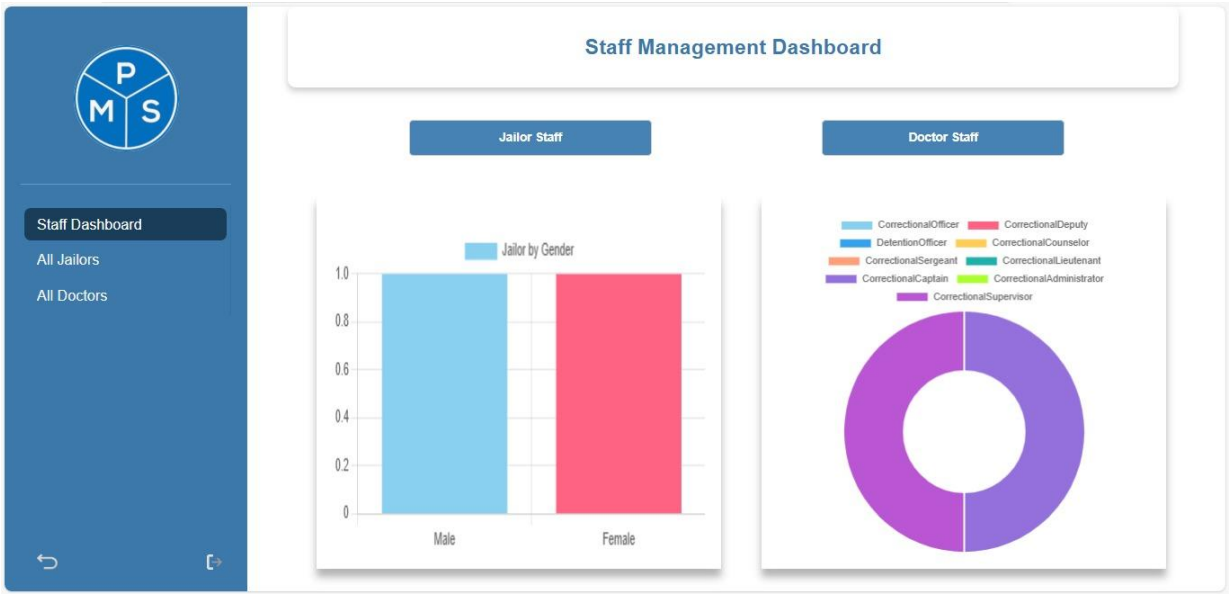


Figure 7 Staff Dashboard

Visitors List

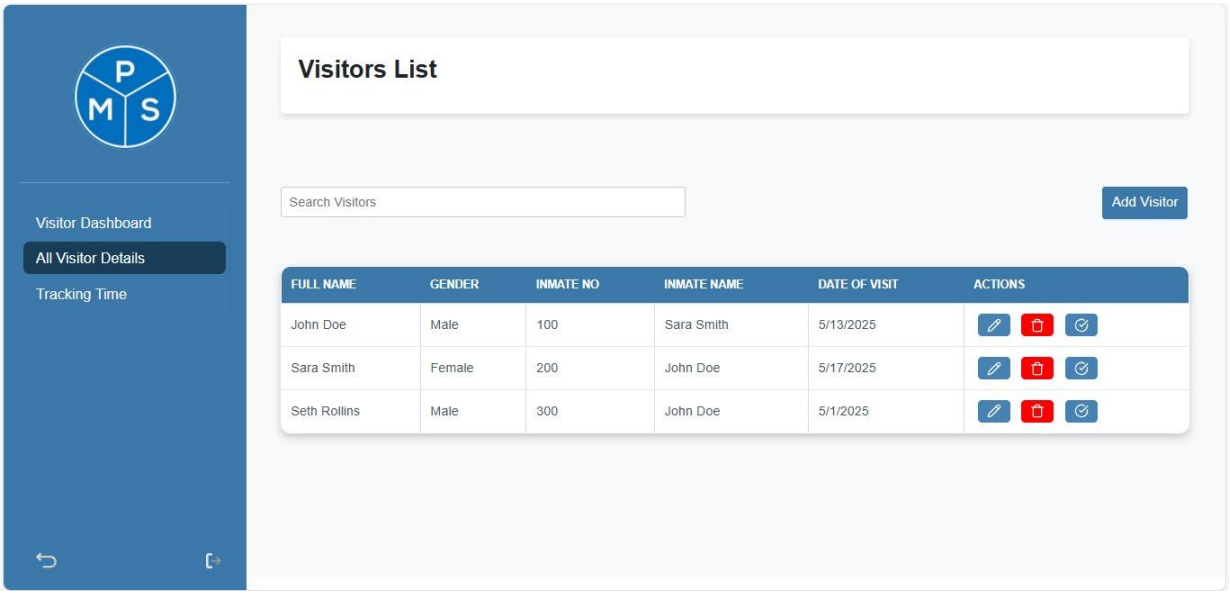


Figure 8 Visitors List

Health Record Form

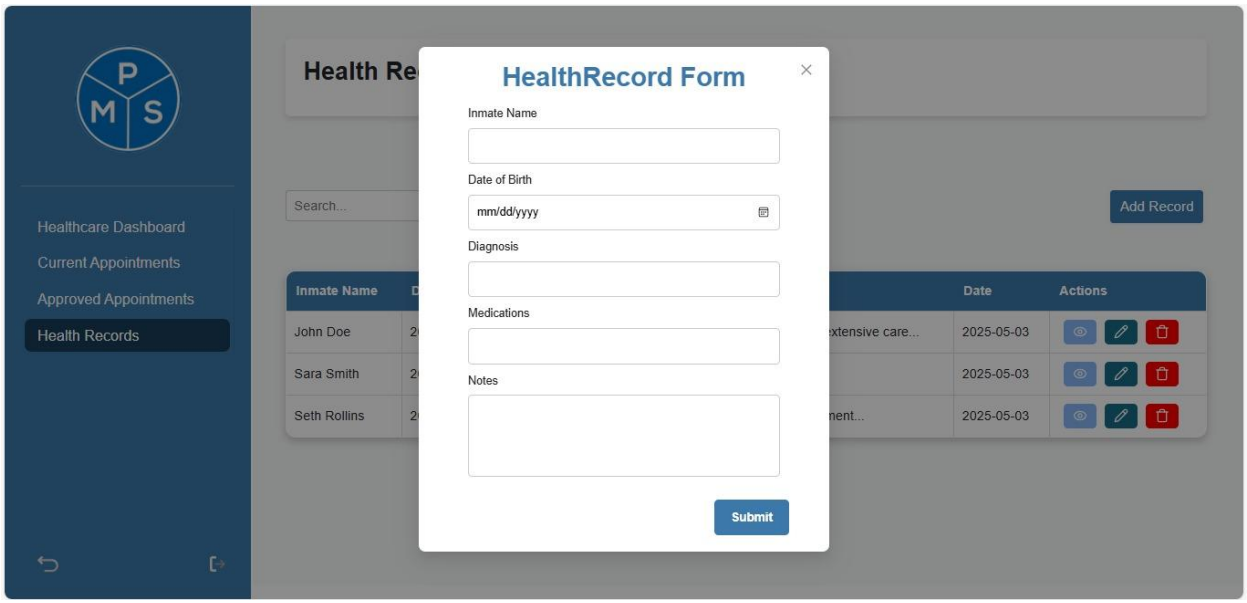


Figure 9 Health Record Form

3.3.2 Screen Objects and Actions

Use Case : Register Prisoner

Form Title:

- Object: "Register Prisoner" (moving text).
- Purpose: Displays the purpose of the form.
- Action: No action; read only.

Input Fields:

ID

- Type : Text input field.
- Purpose : Enter the prisoner's identification number.
- Action : Accepts numeric or alphanumeric input.

Name

- Type : Text input field.
- Purpose : Enter the complete name of the prisoner.
- Action : Accepts alphanumeric input.

Offence

- Type : Text input field.
- Purpose : Enter the offence committed by the prisoner.
- Action : Accepts alphanumeric input.

Date of Birth

- Type : Date-picker.
- Purpose : Choosing the prisoner's birth-date.
- Action : Opens a calendar to choose a date.

Sentence Date

- Type : Date-picker.
- Purpose : This is to state the date of sentencing.
- Action : Opens a calendar to choose a date.

Discharge Date

- Type : Date-picker.
- Purpose : This is to select the expected discharge date.
- Action : Opens a calendar to choose a date.

Gender

- Type: Drop-down Menu.
- Purpose: Select the gender of the prisoner.
- Action: Shows up options (e.g., Male, Female, Other) on the click.

Marital Status

- Type: Drop-down Menu.
- Purpose: Select the prisoner's marital status.
- Action: Shows options (Single, Married, etc.) on the click.

Cell Allocation

- Type : Text input field.
- Purpose : Enter the cell number allocated to the prisoner.
- Action : Accepts numeric or alphanumeric input.

Emergency Contact Name

- Type : Text input field.
- Purpose : Name of prisoner's emergency contact.
- Action : Accepts alphanumeric input.

Emergency Contact Number

- Type : Text input field.
- Purpose : Enter the telephone number of the emergency contact.
- Action : Accepts numeric input.

Emergency Contact Relation

- Type: Input text field
- Purpose: Specify the relationship with the given emergency contact.
- Action: Accepts alphanumeric input. Validate inputs. Submit form if valid.

Register

- Type of Object: Button.
- Intention: Submit this form in order to register into the prison system.
- Actions: Validate inputs. Submit form if valid.

Close Icon

- Object Type: Icon (X at top-right corner).
- Purpose: To close without saving.
- Action: Dismissing pop-up.

Here are some sidebar navigation options:

- View Complaints.
- Manage Medical Requests.
- Manage Visit Requests.
- Schedule Duties.

Purpose : to navigate between the various management actions.
Action : loads the corresponding interface when clicked on it.

Use Case : Request Medical Checkup

Form Title:

- Object: "Request Medical Checkup" (static text).
- Purpose: Describes the intent behind the form.
- Action: No action, as it is read-only.

Input Fields:

Name

- Object Type: Text input field
- Purpose: Entail value of name of prisoner.
- Action: Accepts alphanumeric input.

Prisoner ID

- Object Type: Text input field
- Purpose: Input Unique ID of prisoner.
- Action: Accepts Numeric or Alphanumeric input.

Medical Issue

- Object Type: Text input field
- Purpose: Enter the medical condition or issue experienced
- Action: Accepts alphanumeric input.

Description

- Object Type: Multi-line text area.
- Purpose: For description of the problem.
- Action: Accepts any size alphanumeric input.

Checkbox:

Emergency Request:

- Object Type: Checkbox
- Purpose: To enable the inmate to have the check mark if the request is an emergency.
- Action: Toggle from checked (true) to unchecked (false) state

Buttons:

Reset:

- Object Type: Button
- Purpose: Clears all field inputs so that the inputs may be reset in the form.
- Action: This will revert all fields to their default values.

Submit:

- Object Type: Button
- Purpose: Where it submits the form for processing.
- Action: Validate all inputs on the form. Then if validated, send the request for further processing into the system. If validation fails, then prompts for an error message.

Close Icon:

- Object Type: Icon (X in the top-right corner).
- Purpose: It closes a medical request form.
- Action: It dismisses the modal without saving any changes.

Sidebar Navigation:

Options:

- File Complaint
- Request Medical Checkup (highlighted).
- Request Visitor.
- Request Work Assignment.

Purpose: It allows shifting from one action to another for the inmate.
Action: Clicking it will load up the relevant interface.

Table of Requests:

Columns:

- Request No.: Displays a unique identifier for request.
- Status: Shows the status of each request.

Purpose: It gives an overview of the history of a prisoner's medical requests.
Action: Read-only show; no interaction.

Actions:

Fill the Form:

- User Types with Name, Prisoner ID, Medical Issue, and Description fill all the fields.
- Optional: Checks the Emergency Request box, if applicable.

Resetting the Form: Clicking the Reset button clears all fields to default (empty).

Submitting the Request: Clicking Submit validates the entries as follows:

- All required fields filled up. If the request is valid, it submits and updates the table.
- In case it is not valid, it gives an error message to prompt the user for correction.

Closing the Form: The modal is dismissed by the close icon and back into dashboard view.

3.6 Behavioural Model

Interaction Diagrams: (sequence diagrams)

UC 1 : Login

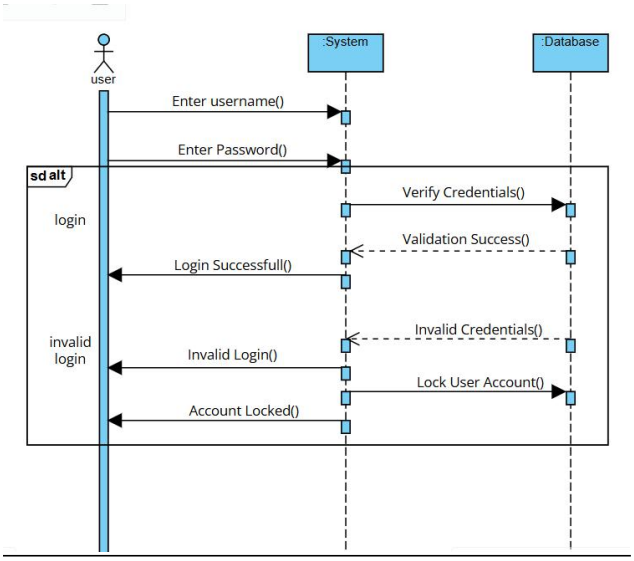


Figure 10 Sequence Diagram for Use Case Login

UC 2 : Register Prisoner

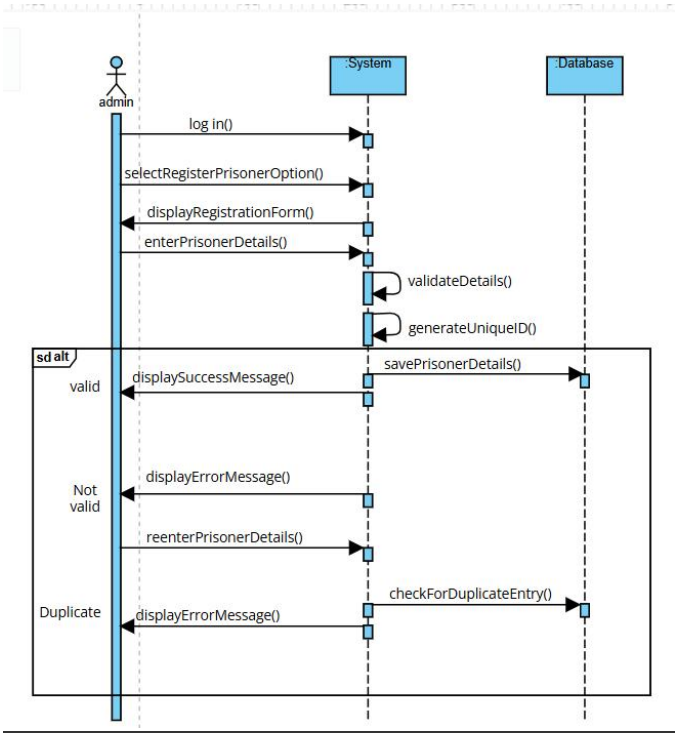


Figure 11 Sequence Diagram for Use Case Register Prisoner

UC 3 : View Prisoner Record

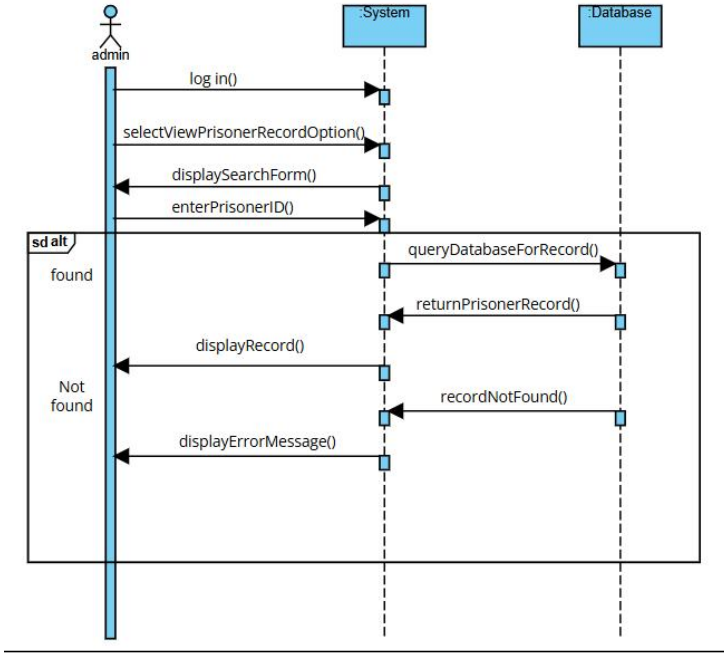


Figure 12 Sequence Diagram for Use Case View Prisoner Record

UC 4 : Update Prisoner Record

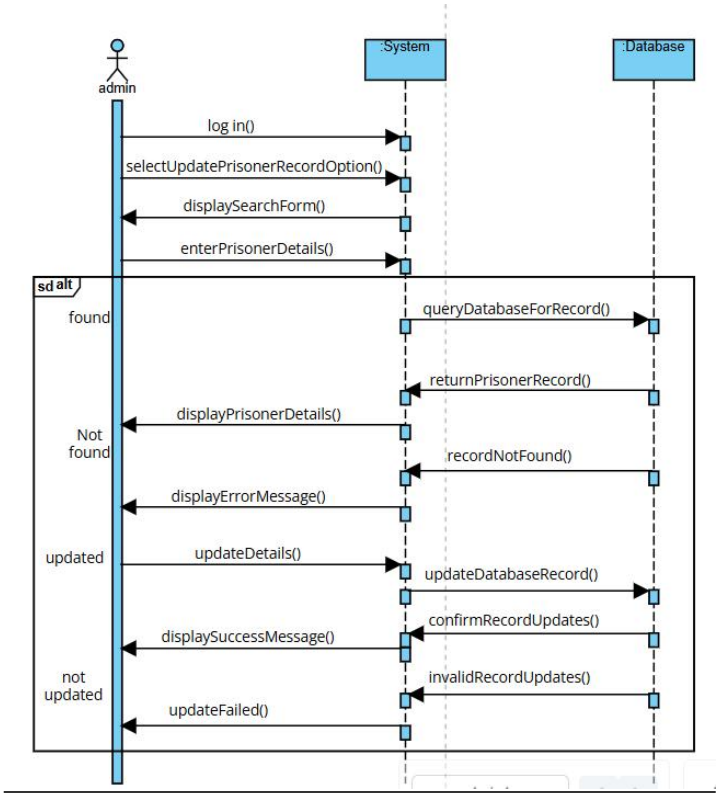


Figure 13 Sequence Diagram for Use Case Update Prisoner Record

UC 5 : Remove Prisoner Record

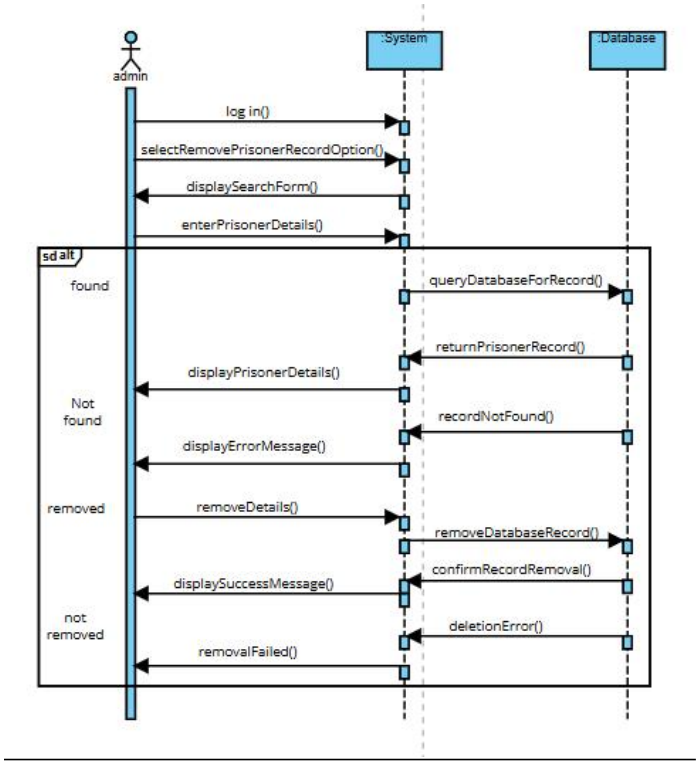


Figure 14 Sequence Diagram for Use Case Remove Prisoner Record

UC 6 : Allocate Cells

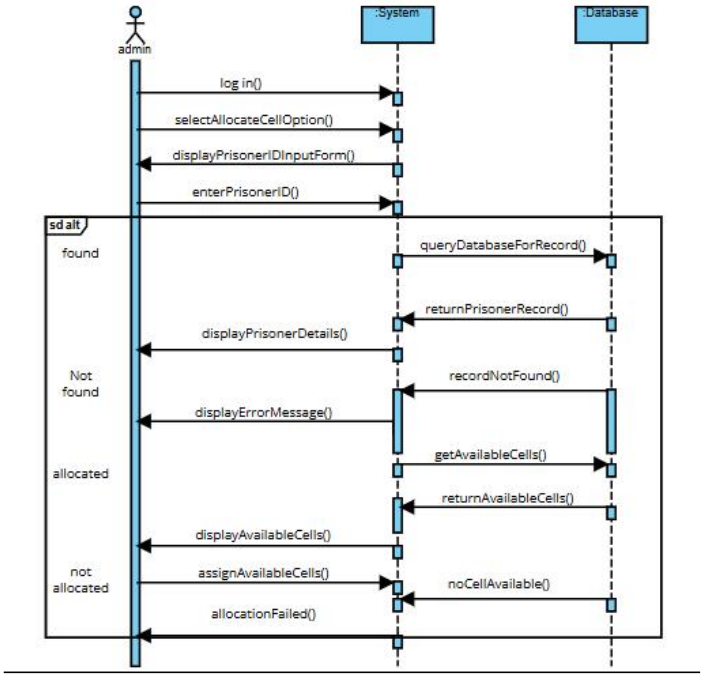


Figure 15 Sequence Diagram for Use Case Allocate Cells

UC 7 : Schedule Duties

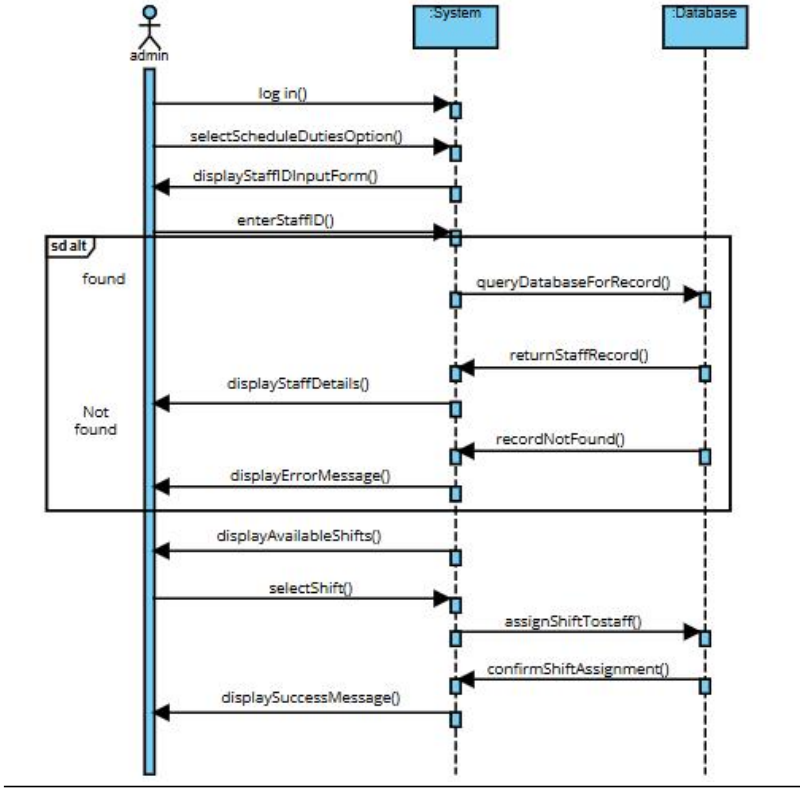


Figure 16 Sequence Diagram for Use Case Schedule Duties

UC 8 : View Complaints

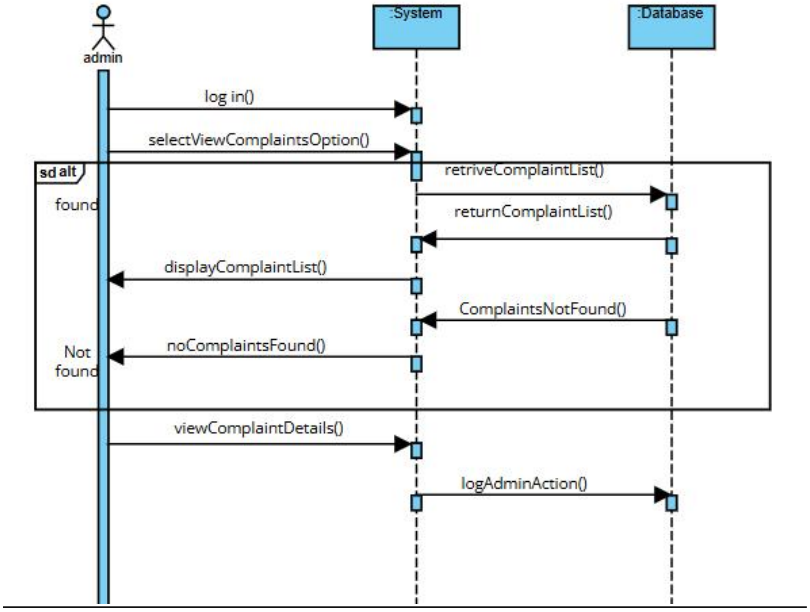


Figure 17 Sequence Diagram for Use Case View Complaints

UC 9 : Manage Visit Requests

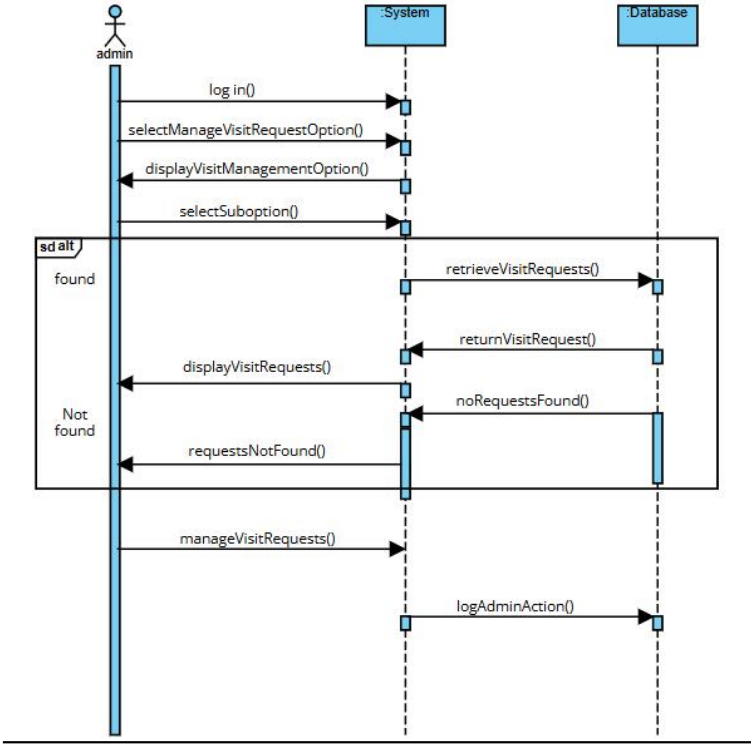


Figure 18 Sequence Diagram for Use Case Manage View Requests

UC 10 : Manage Prisoner Visit Requests

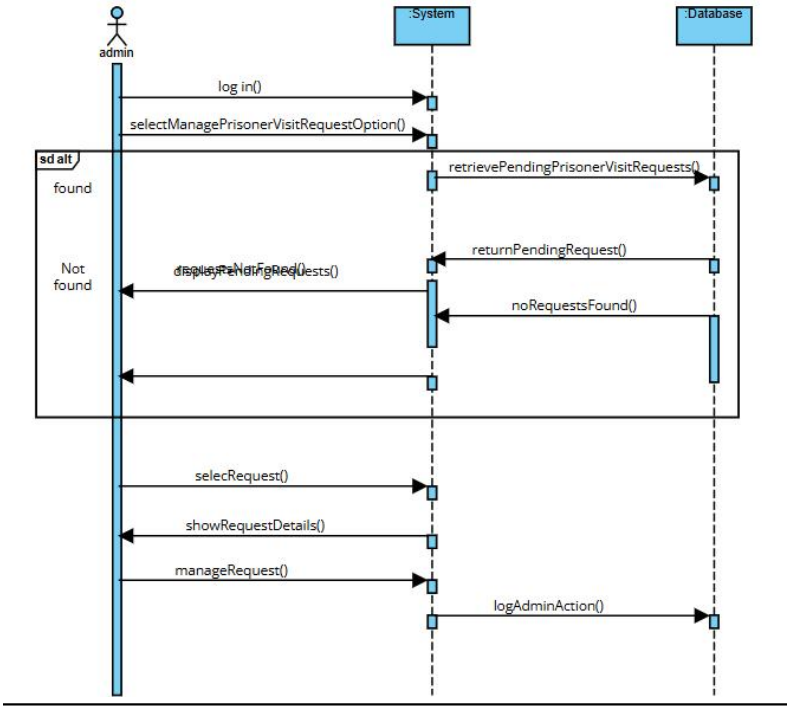


Figure 19 Sequence Diagram for Use Case Manage Prisoner Visit Requests

UC 11 : Manage Visitor Visit Requests

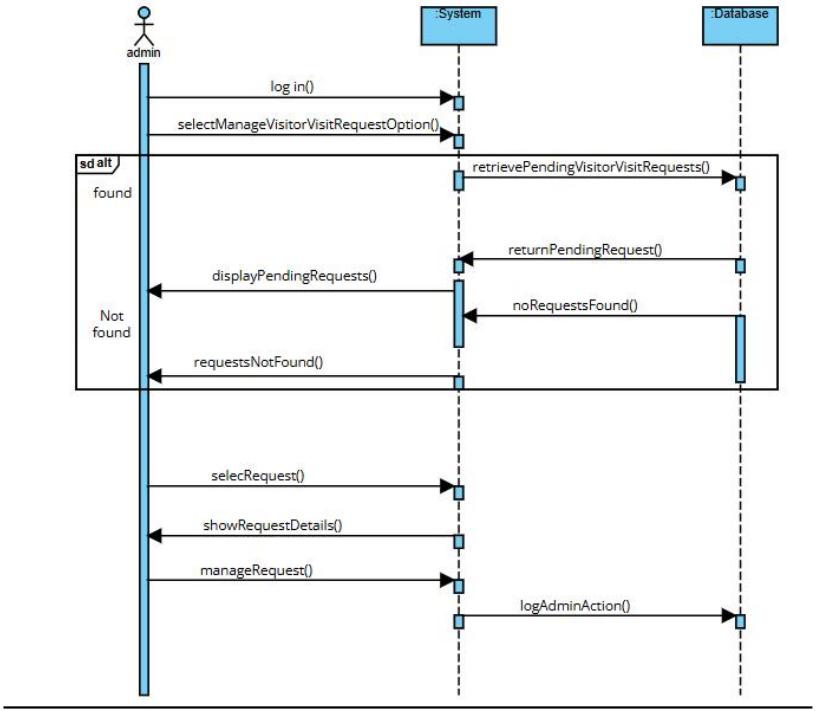


Figure 20 Sequence Diagram for Use Case Manage Visitor Visit Requests

UC 12 : Manage Medical Requests

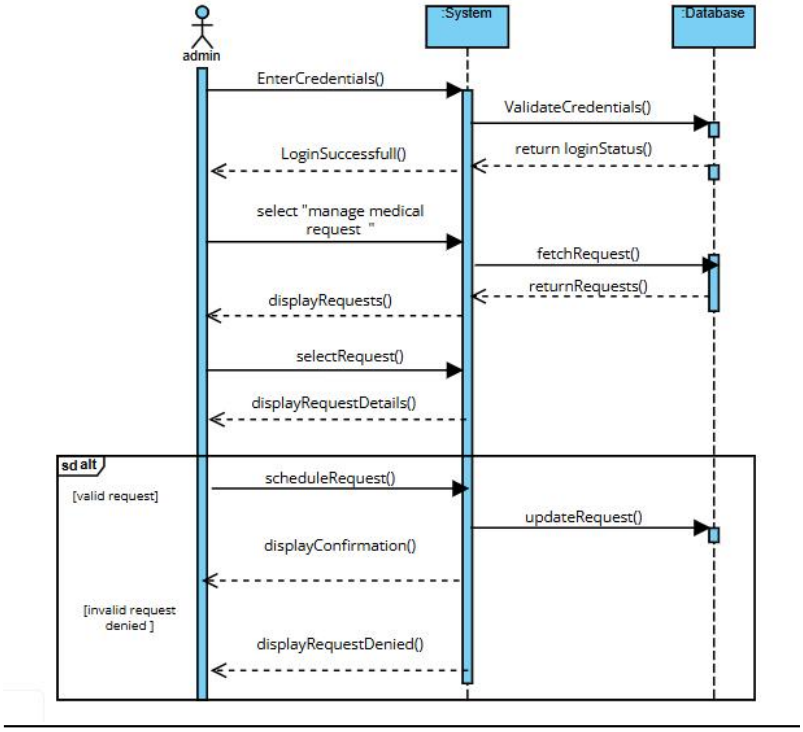


Figure 21 Sequence Diagram for Use Case Manage Medical Requests

UC 13 : View Duties Schedule

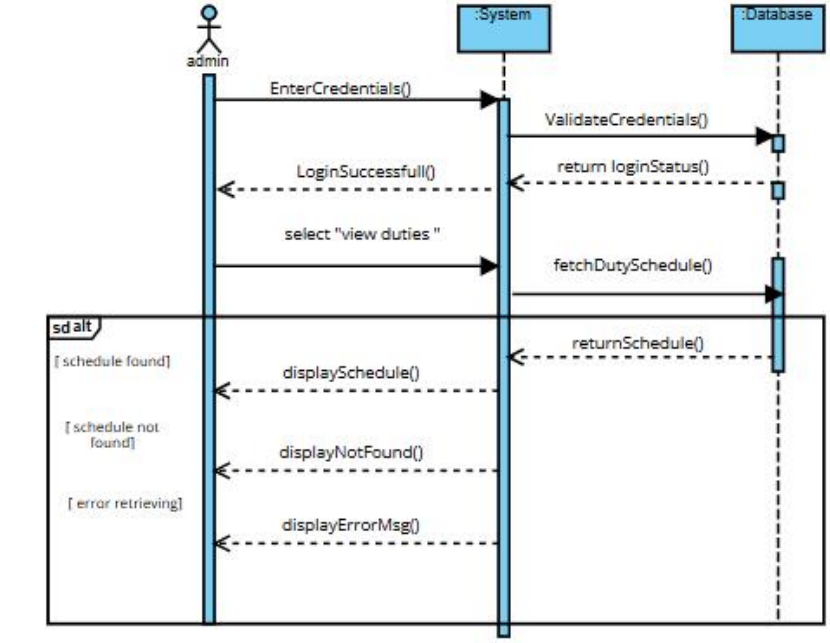


Figure 22 Sequence Diagram for Use Case View Duties Schedule

UC 14 : Assign Work

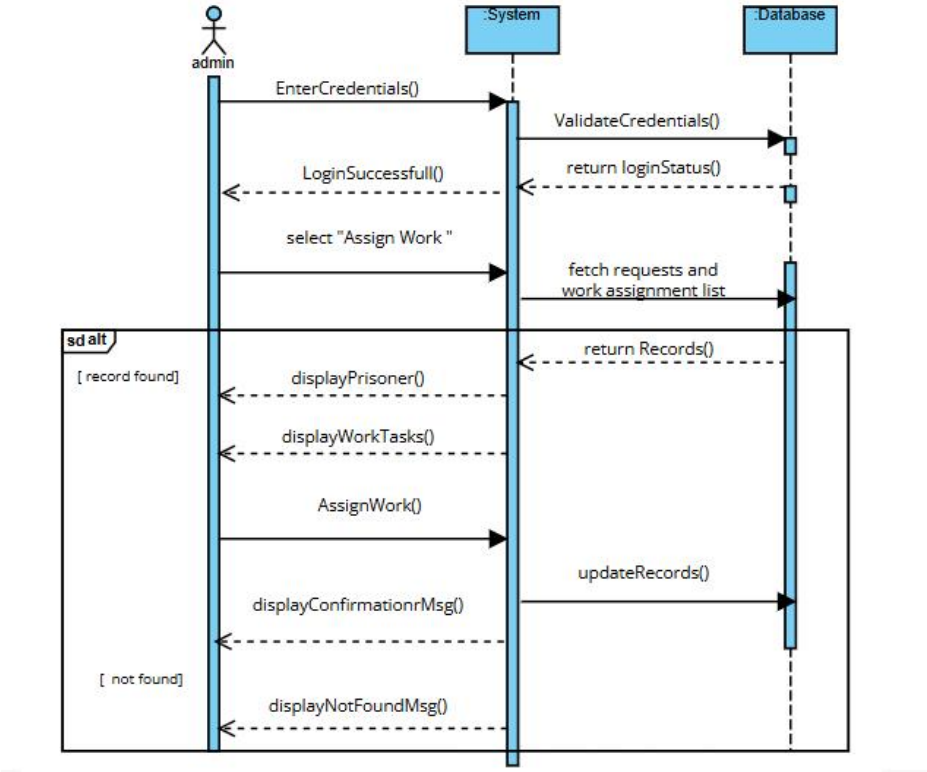


Figure 23 Sequence Diagram for Use Case Assign Work

UC 15 : Permit Medical Requests

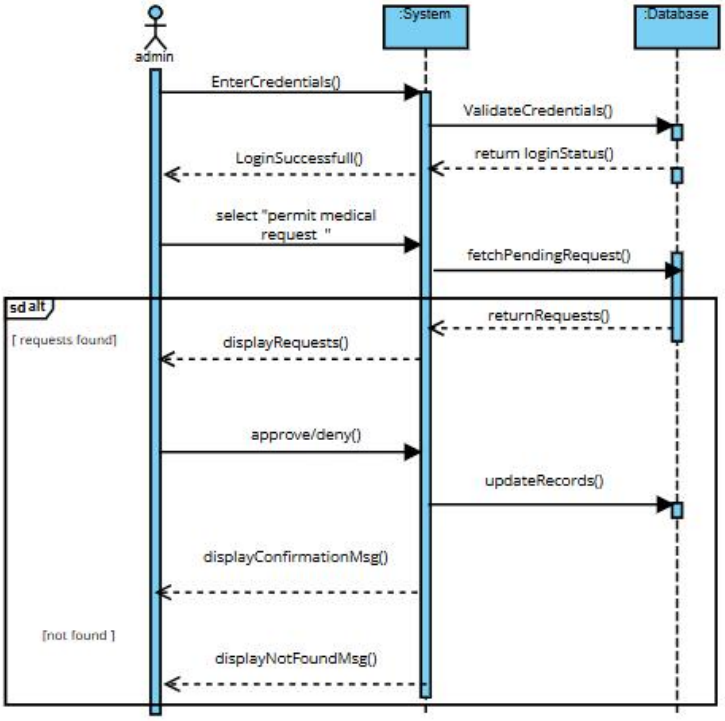


Figure 24 Sequence Diagram for Use Case Permit Medical Requests

UC 16 : Request Visitor

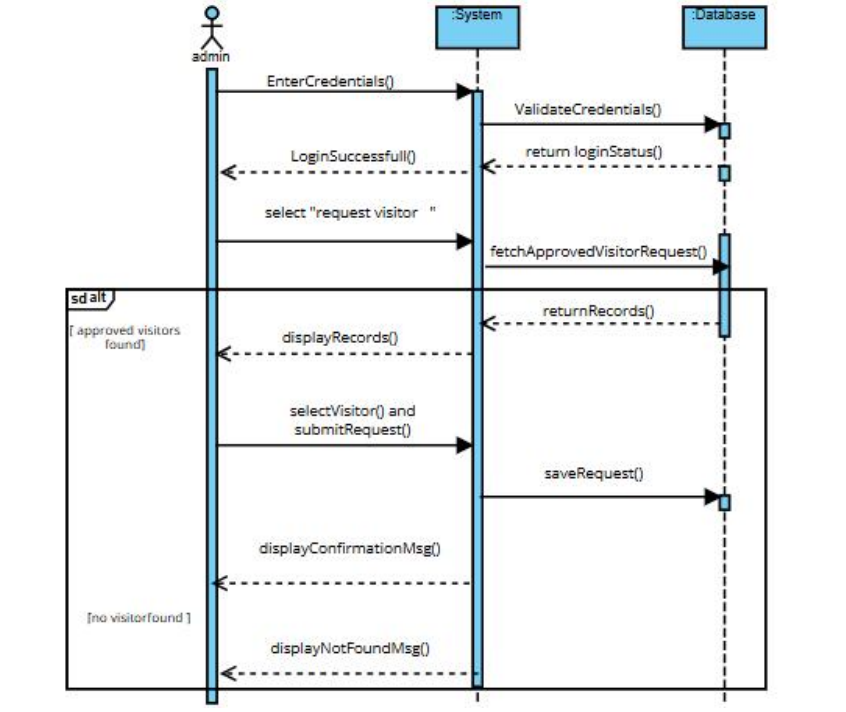


Figure 25 Sequence Diagram for Use Case Request Visitor

UC 17: Request Work Assignment

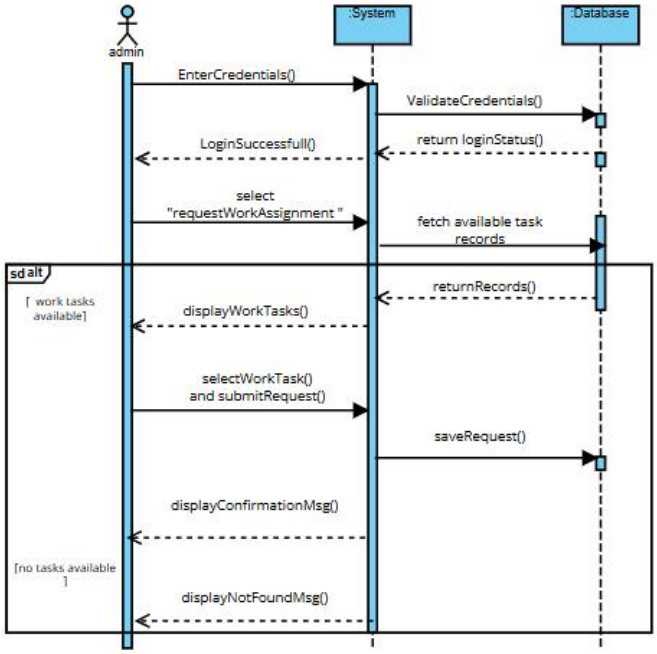


Figure 26 Sequence Diagram for Use Case Request Work Assignment

UC 18 : Request Medical Checkup

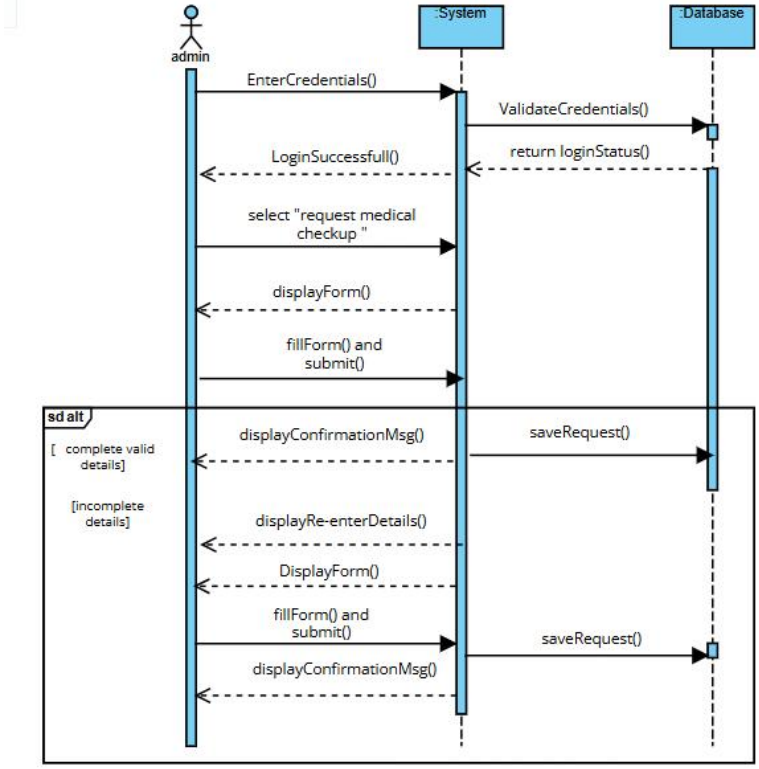


Figure 27 Sequence Diagram for Use Case Request Medical Checkup

UC 19 : File Complaints

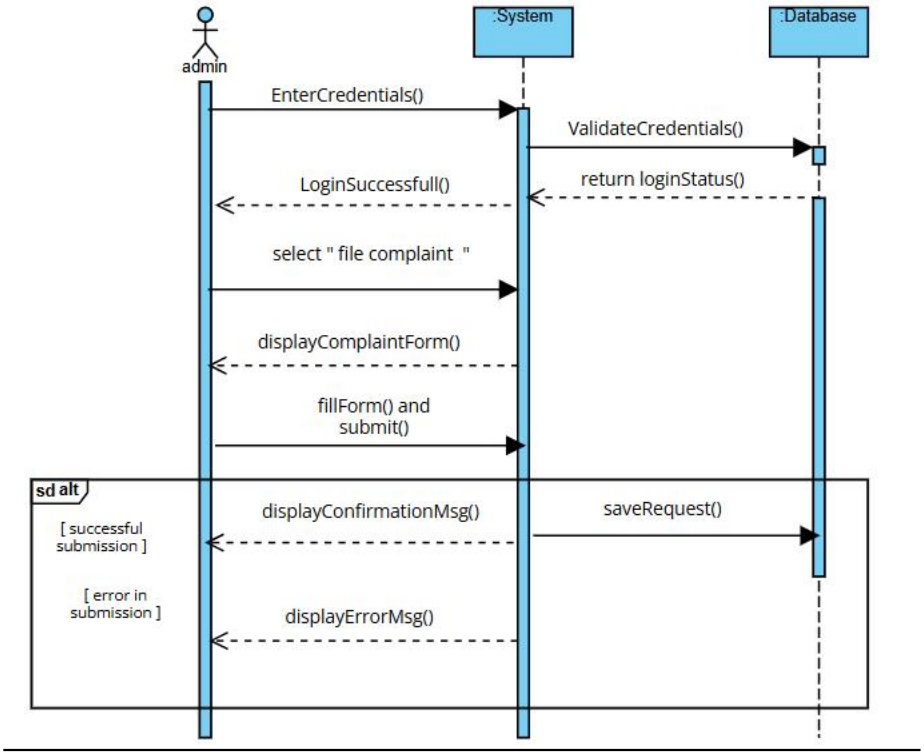


Figure 28 Sequence Diagram for Use Case File Complaints

UC 20 : Request Visit

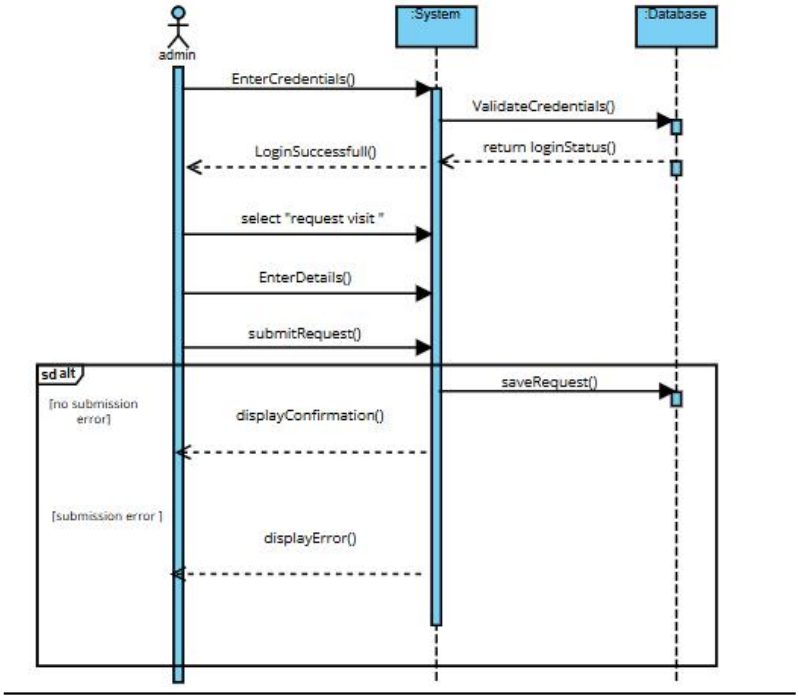


Figure 29 Sequence Diagram for Use Case Request Visit

UC 21 : View Request Status

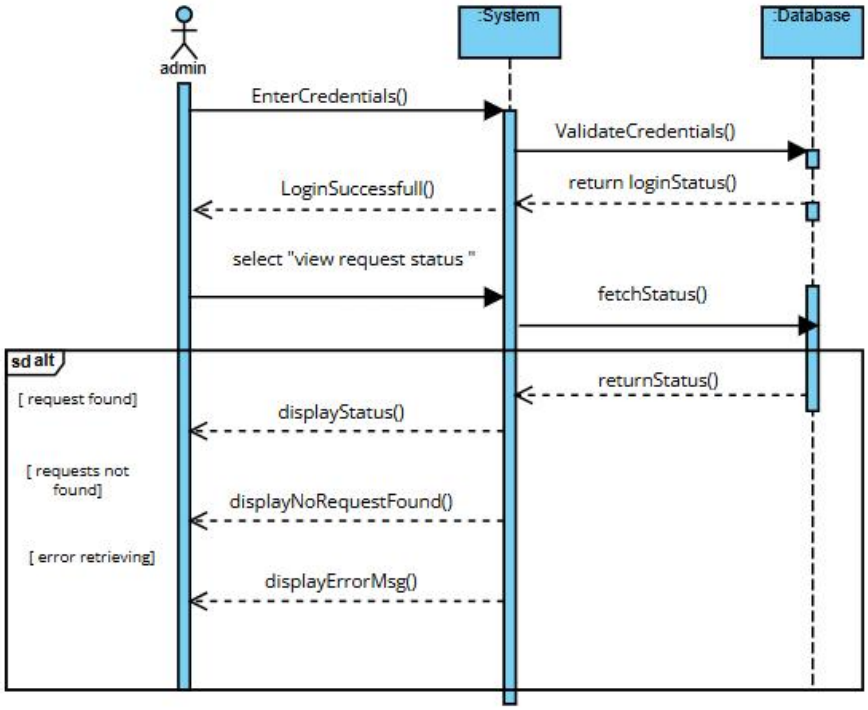


Figure 30 Sequence Diagram for Use Case View Request Status

State Diagrams:

UC 1 :Login

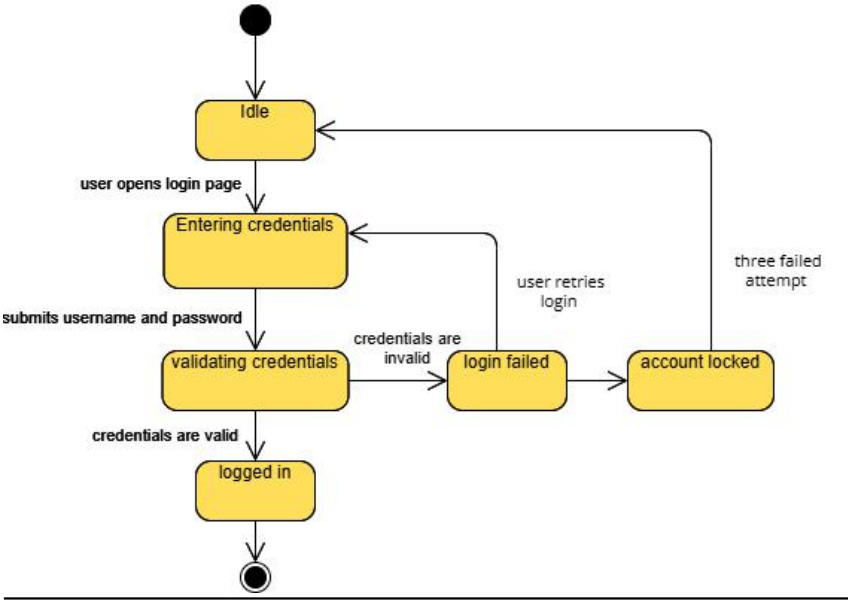


Figure 31 State Transition Diagram for Use Case Login

UC 2 : Register Prisoner

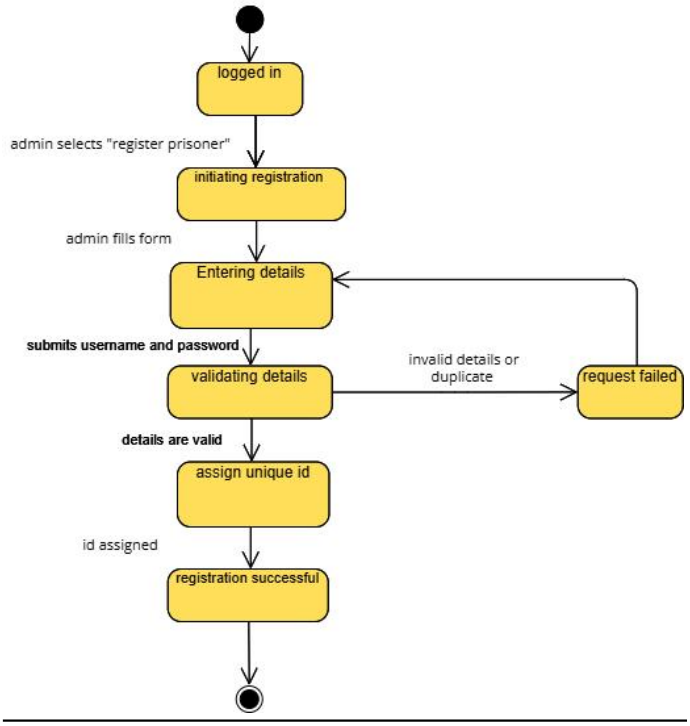


Figure 32 State Transition Diagram for Use Case Register Prisoner

UC 3 : View Prisoner Record

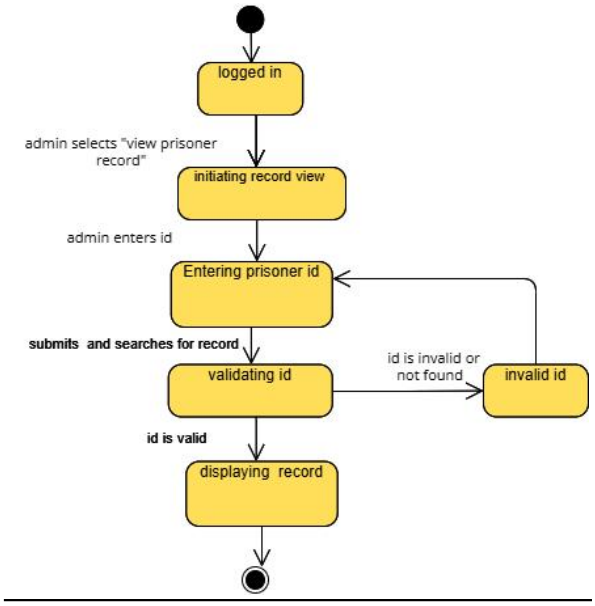


Figure 33 State Transition Diagram for Use Case View Prisoner Record

UC 4 : Update Prisoner Record

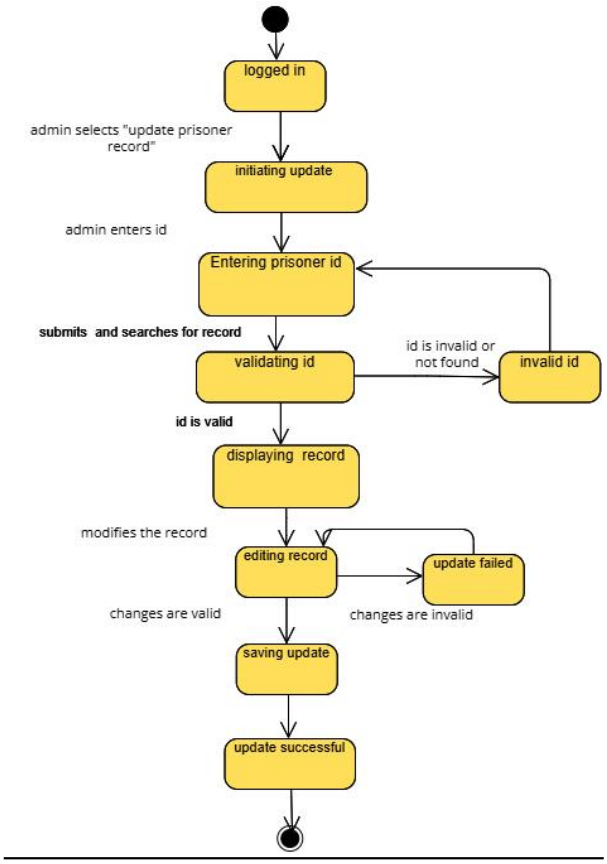


Figure 34 State Transition Diagram for Use Case Update Prisoner Record

UC 5 : Remove Prisoner Record

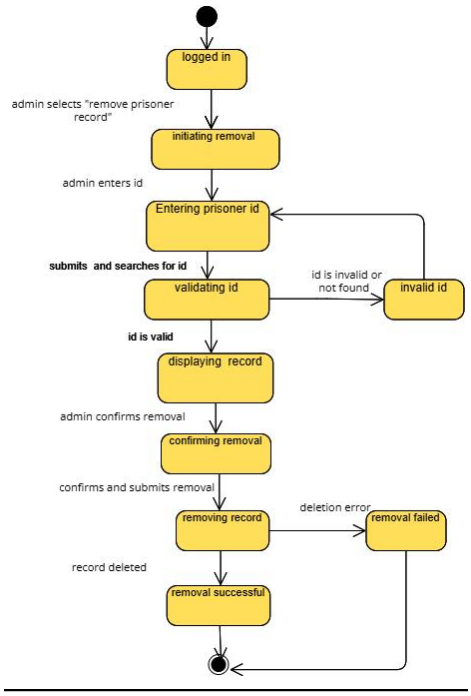


Figure 35 State Transition Diagram for Use Case Remove Prisoner Record

UC 6 : Allocate Cells

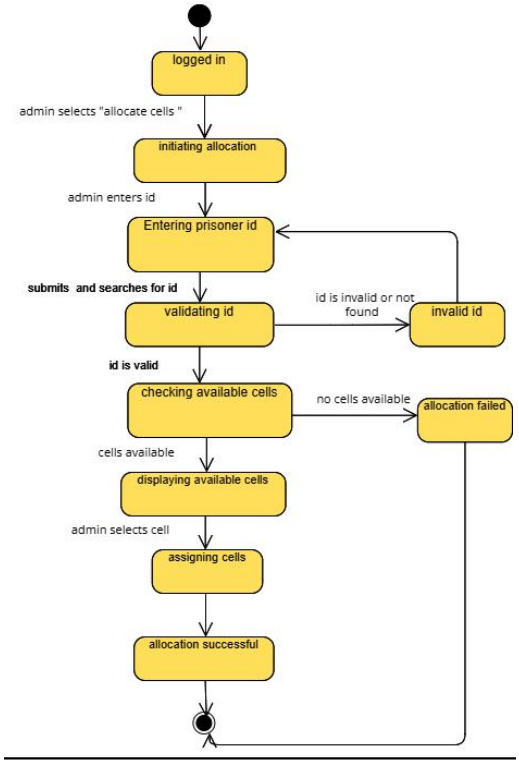


Figure 36 State Transition Diagram for Use Case Allocate Cells

UC 7 : Schedule Duties

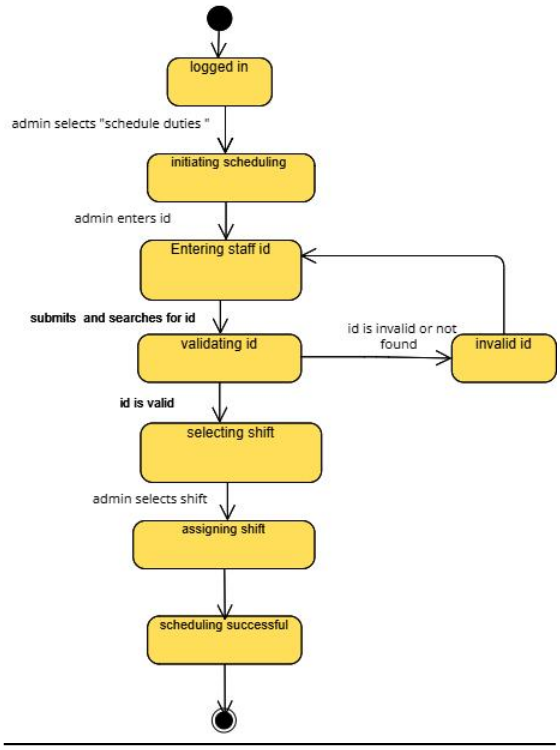


Figure 37 State Transition Diagram for Use Case Schedule Duties

UC 8 : View Complaints

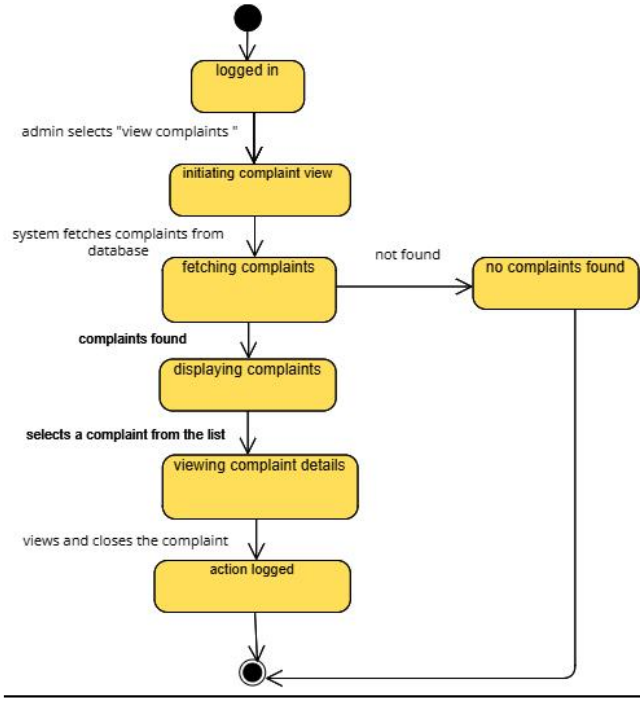


Figure 38 State Transition Diagram for Use Case View Complaints

UC 9 : Manage Visit Requests

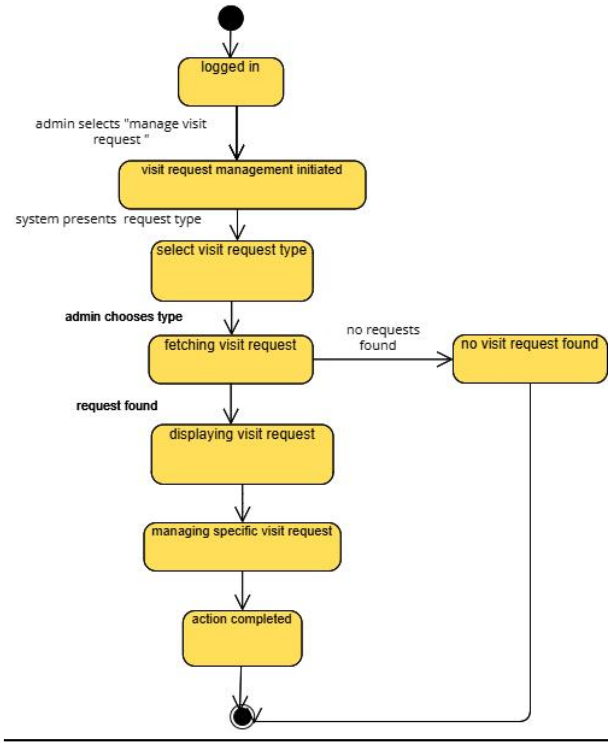


Figure 39 State Transition Diagram for Use Case Manage Visit Requests

UC 10 : Manage Prisoner Visit Requests

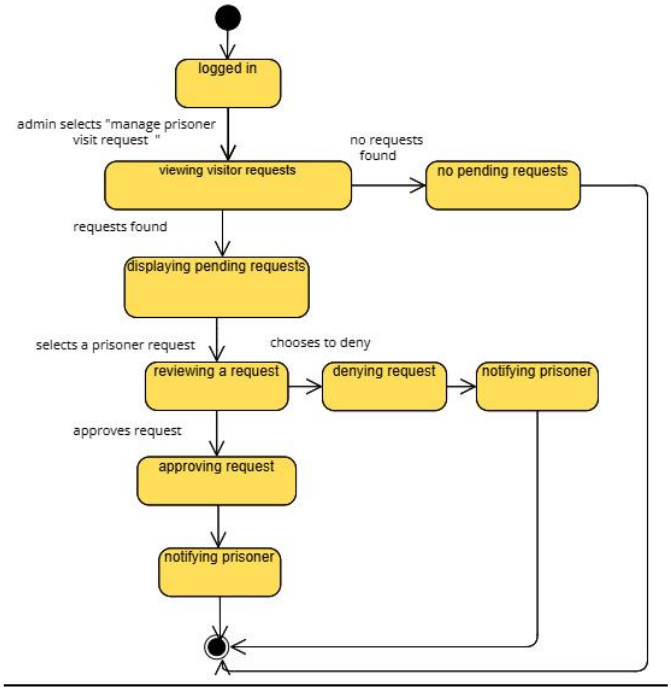


Figure 40 State Transition Diagram for Use Case Manage Prisoner Visit Requests

UC 11 : Manage Visitor Visit Requests

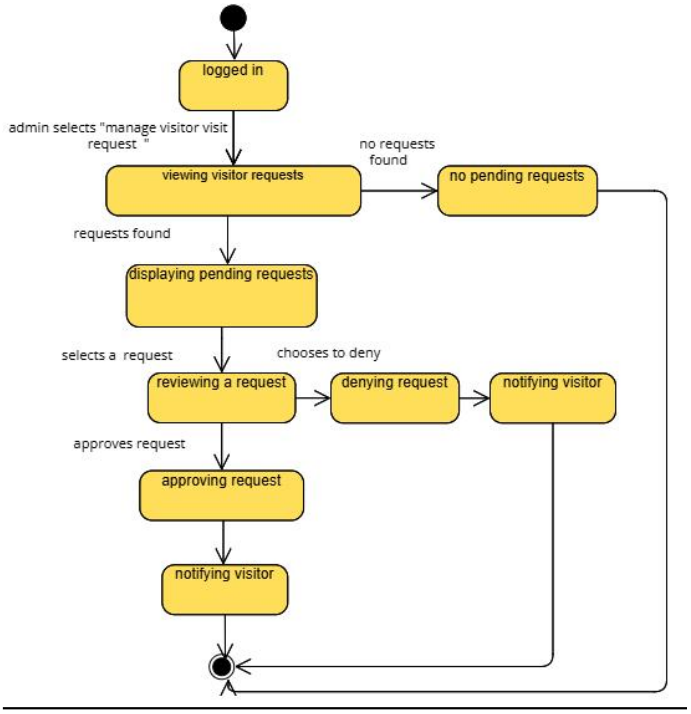


Figure 41 State Transition Diagram for Use Case Manage Visitor Visit Requests

UC 12 : Manage Medical Requests

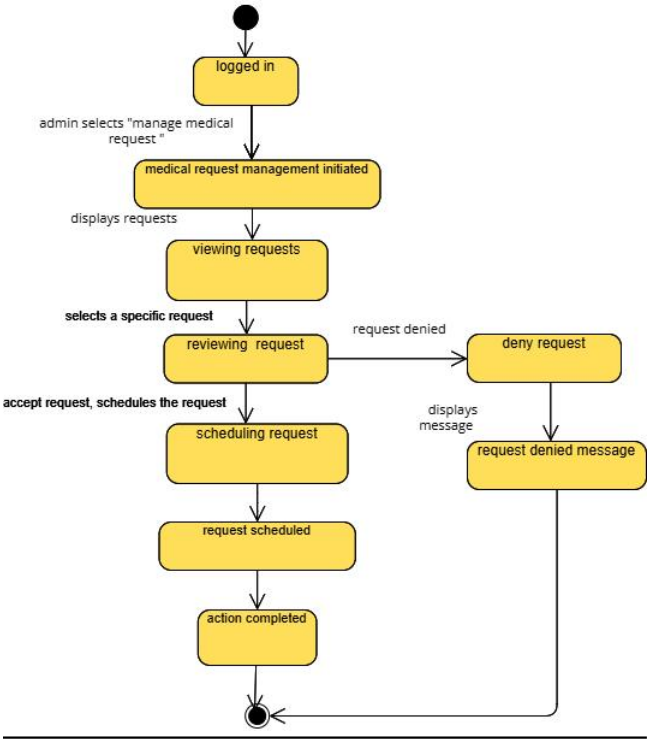


Figure 42 State Transition Diagram for Use Case Manage Medical Requests

UC 13 : View Duties Schedule

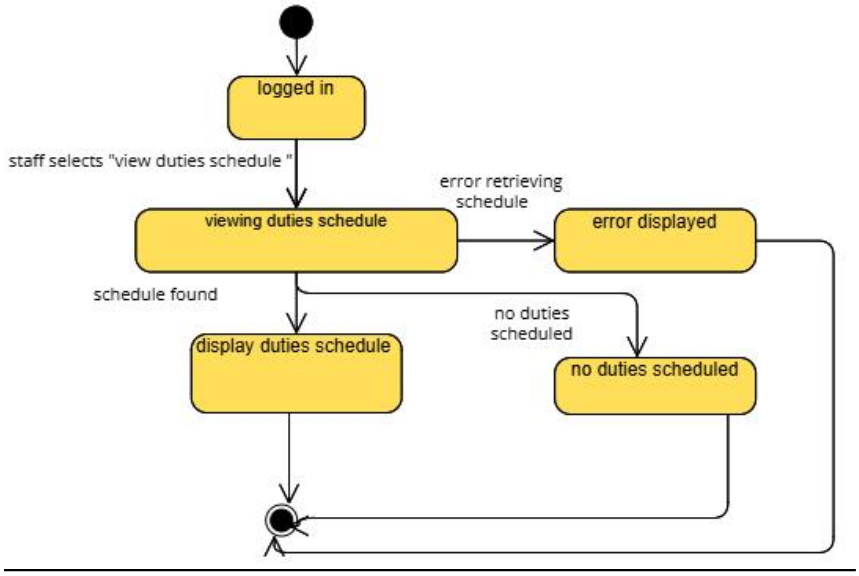


Figure 43 State Transition Diagram for Use Case View Duties Schedule

UC 14 : Assign Work

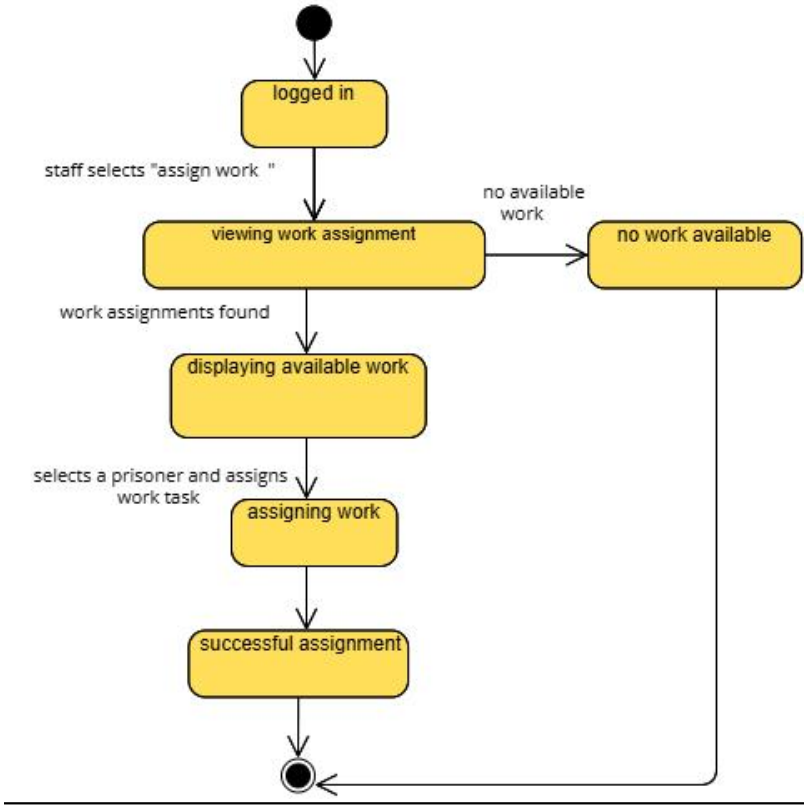


Figure 44 State Transition Diagram for Use Case Assign Work

UC 15 : Permit Medical Requests

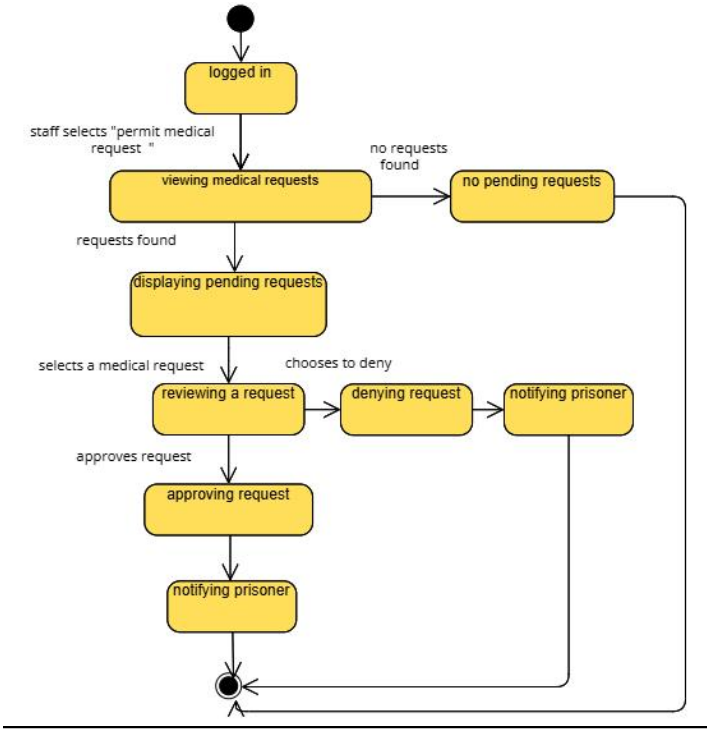


Figure 45 State Transition Diagram for Use Case Permit Medical Requests

UC 16 : Request Visitor

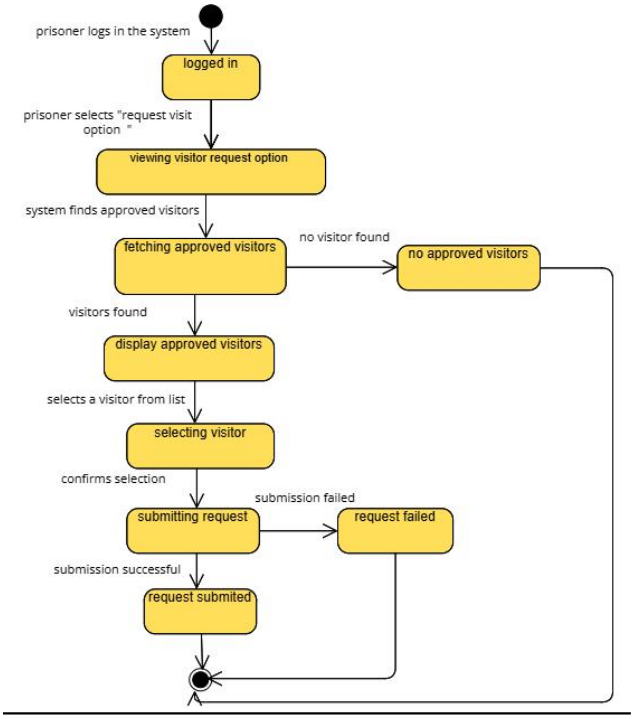


Figure 46 State Transition Diagram for Use Case Request Visitor

UC 17 : Request Work Assignment

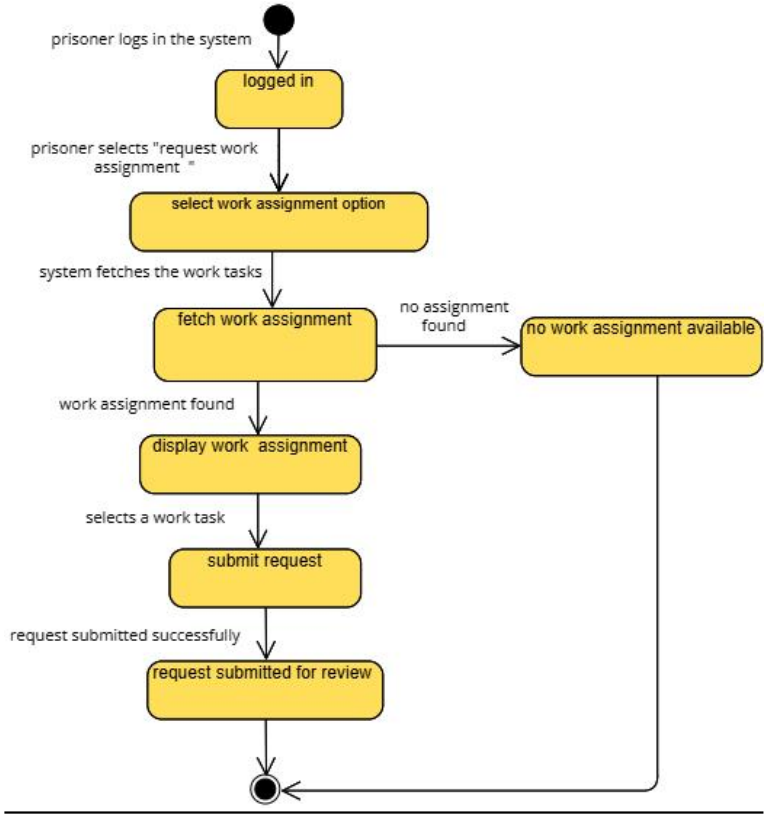


Figure 47 State Transition Diagram for Use Case Request Work Assignment

UC 18: Request Medical Checkup

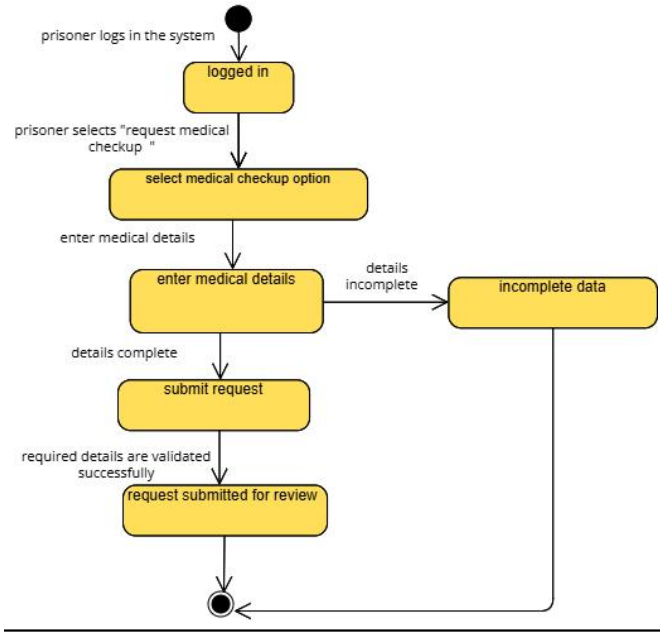


Figure 48 State Transition Diagram for Use Case Request Medical Checkup

UC 19 : File Complaints

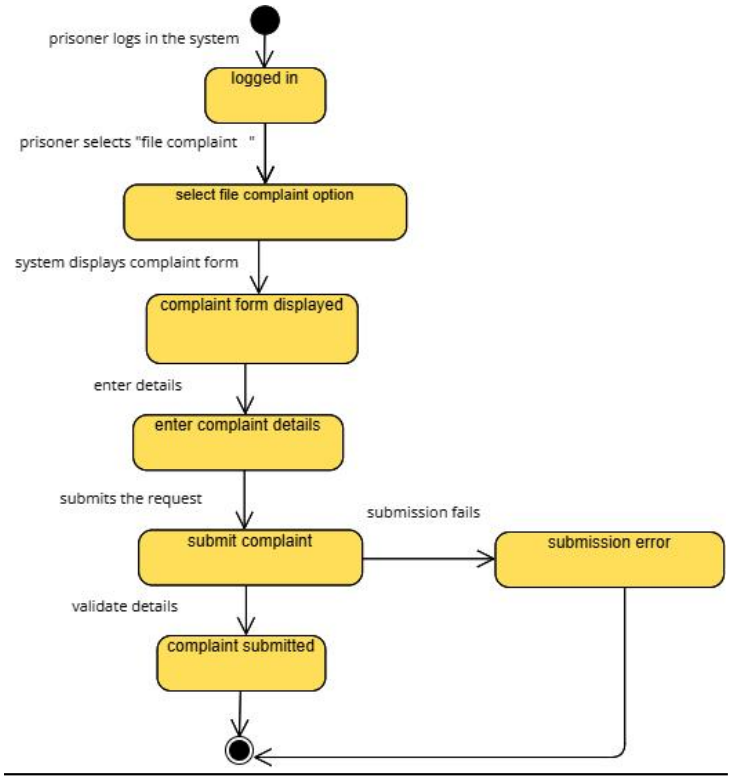


Figure 49 State Transition Diagram for Use Case File Complaints

UC 20 : Request Visit

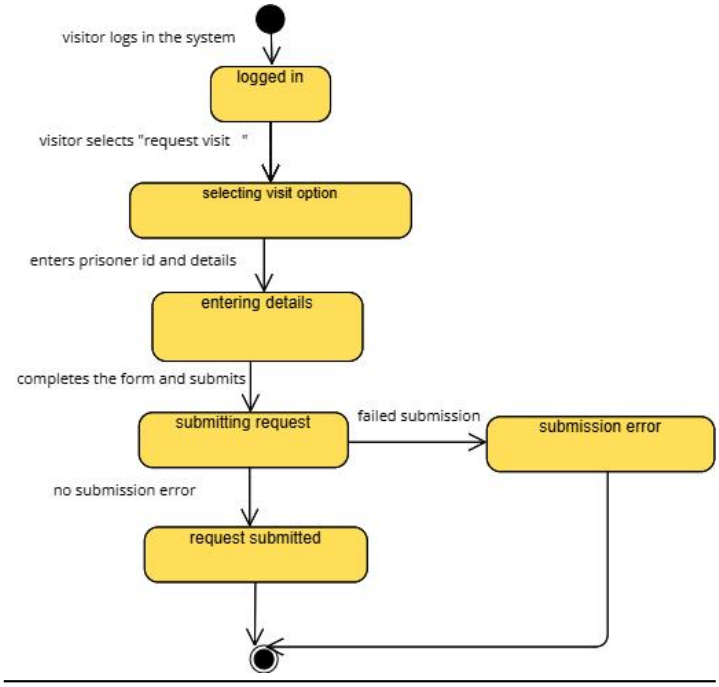


Figure 50 State Transition Diagram for Use Case Request Visit

UC 21 : View Request Status

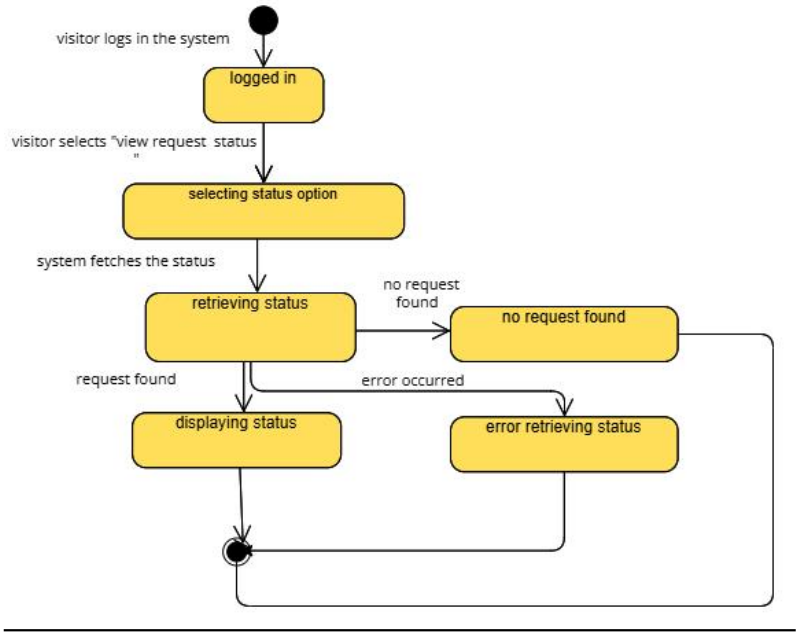


Figure 51 State Transition Diagram for Use Case View Request Status

3.7 Design Decisions

1. Object-Oriented Design Pattern

Choice: Object-Oriented Programming (OOP)

Reason:

- The system consists of various entities like Prisoner, Visitor, Administrator, etc., which can be easily modeled as classes with their attributes and behaviors.
- OOP provides features like encapsulation, inheritance and polymorphism making it easy for maintenance and extension of the system.
- Relationships exist among the entities such as Administrator manages Prisoner, which are so naturally represented by associations, inheritance and compositions in UML diagrams.

Advantages:

- Scalability- Adding new entities for example "ParoleOfficer" or "InmateActivityLog" would be achievable without disrupting the already existing code.
- Reusability- Shared functionality (like request handling) could be abstracted into base classes.
- Maintainability: Encapsulation ensures data security and modularity.

2. Database Design and Normalization

Choice: Third Normal Form (3NF)

Reason:

- Ensures that all data is stored in a non-redundantly organized manner by regrouping attributes into tables, as much as possible, and assigning them with a minimum of duplication value.
- Creates a degree of consistency in data, which means fewer anomalies and optimal storage.
- Example: Separate tables like Prisoners, VisitorRequests, and Complaints are linked using foreign keys.

Advantages:

- **Data Integrity**: Ensures no redundancy in prisoner details across multiple tables.
- **Query Optimization:** Smaller tables help speed up queries.
- **Maintenance:** Changing schema is not cumbersome.
- **Trade off:** Most joins are needed when looking into 3NF; however, the loss of redundancy and consistency from this more than makes up for the joins required.

3. Algorithmic Approach

Selection: CRUD Operations with Query Optimization

Reason:

- The system is only supplementing CRUD (Create, Read, Update, Delete) operations for the management of entities like prisoners, visitors, and staff.
- The focus will be on those indexing keys such as prisonerID, visitorID, and requestID to improve query performance. Query optimizations will ensure that smooth operation with large sets of data.
- Examples: Efficiently fetching prisoner details by indexing prisonerID. Using joins to fetch related data such as prisoner complaints and medical requests in a single query.

Benefits: Fast data retrieval with simple implementation. Performance under high load.

4. Design Patterns Selection: MVC (Model-View-Controller)

- **Pattern Reason:** The application may be separated into three components-M;
- **Model**: It is representative of the data and business logic (for example class like Prisoner and Complaint).
- **View:** The user interface representation (for example dashboards for administrators, staff, and visitors).
- **Controller:** handles input and updates<|image_sentinel|>the model and its view (for example, handling CRUD operations). Separation easily modulates so that the system can become an easy candidate for its testing and maintenance.
- **Benefits:**
- **Flexible:** The change in UI (View) does not affect the business logic (Model).
- **Testability:** Logic can be tested independently of UI.

5. Handling the Concurrency Handling

Selection: Optimistic Concurrency Control

Rationale:

- The system is likely to be accessed and updated by multiple users, such as the staff editing Duty Schedules and visitors submitting visit requests. One example: it waits until two administrators have finished updating a prisoner's record for it to check the version of that record before committing changes.
- An Ideal Choice for Avoiding Unnecessary Locks Which Improve Performance.
- Best for systems which often perform more read-heavy operations than writes.

6.User Authentication and Security:

Role-Based Access Control (RBAC)

Reason:

- Allows specific permission and access for different roles like Administrator, Prison Staff and Visitor. Only administrators were proven to allocate cells or remove prisoners, hence role-based control would restrict unauthorized access.

Advantages:

- It prevents unauthorized access to data; it offers a very simple way to manage permissions in defining roles.

7.Tech Stack:

- **Choice:** Language of Programming-Python because this would have a strong library ecosystem and is relatively easy constructing back-end systems.
- **DBMS:** MYSQL because the language is highly robust in supporting relational data and is highly scalable.
- **Framework:** Django as this built-in support makes rapid development with O.R.M. and authentication.
- **Front-end Framework:** React for delivering interactive user interfaces.

8. Exception Handling and Validation

Selection: Centralized Error Handling and Input Validation

Justification:

- The approach should ensure the system will elegantly handle invalid inputs, for example an invalid visitorID during the course of a request.
- Make meaning of user feedback and log errors into the system for debugging.

Implementation:

- Import try and catch blocks during handling of the database operations.
- All inputs have been validated at the controller level and passed on to the processing.

Advantages:

- Preventing the system from crashing due to unexpected errors.
- Improving users' experience.

3.8 Summary

Architectural Design :

This system is a multi-layered architecture that observes the principles of MVC. The layered separation allows the independent development and maintenance of each of the layers.

- **Presentation Layer:** Offers user interfaces for administrators, staff, prisoners, and visitors. The user interfaces are easy to use and accessible.
- **Business Logic Layer:** Deals with core functionalities including prisoner registration, complaint resolution, medical request processing, and visitor scheduling.
- **Database Layer:** Relates and stores data. Data is normalized to reduce redundancy and ensure integrity.

Object-Oriented Design :

The system models prisoners, visitors, and staff as classes that have attributes and methods, thus allowing for:

- **Encapsulation** - ensures data is secured from direct access on the class attributes
- **Inclusion or inheritance** - which increases reusability and hence facilitates the addition of new roles such as the parole officers
- **Polymorphism** - simplifies the adding of new features since any new methods can be overridden
- **Database Normalization:** Having the tables under Third Normal Form (3NF) to reduce redundancy and guard the integrity of data.
- **Concurrency Control:** An optimistic concurrency for simultaneous access to optimize the performance of the system during peak usage.

Security Measures :

- **RBAC:** Limits actions introduced according to the user's role (e.g., only the administrator can assign cells).
- **Encryption:** It provides the effective storage and transmission of data that are sensitive and with security in mind on the cloud.
- **Audit Trails:** Various activities that a user does can be captured and the data can be used to verify accountability and transparency.

Refinements and Trade-offs :

Design decisions were refined to balance performance, scalability, and security:

- **Concurrency Management:** Optimistic control avoids unnecessary locks, making the system more responsive to read-heavy operations.
- **Security vs. Convenience:** Strong authentication mechanisms are built to ensure security even if they are less convenient-e.g., MFA.

Database Normalization: 3NF is a design decision that tends to increase query complexity with benefits of data consistency outweighing performance costs.

Chapter 4

Implementation

4. Implementation

4.1 Algorithm

Table 26 Algorithm User Authentication

Algorithm 1 User Authentication
Input: username, password
Output: Access Granted or Error Message
Time Complexity: 0(1)
1: Initialize error as empty 2: On form submission: 3: If username OR password is empty: 4: Set error to "Username and password are required!" 5: Exit algorithm 6: Else 7: Create a credentials object with username and password 8: Send POST request to server at endpoint "/users/login" with credentials 9: Await server response 10: If response indicates success: 11: Save the received token to local storage 12: Redirect user to Admin Dashboard 13: Else 14: Set error to "Invalid username or password!" 15: Handle any network errors: 16: Set error to "Login failed. Please try again."

Table 27 Algorithm Register Prisoner

Algorithm 2 Register Prisoner Component
Input: : Prisoner Details (image, fullname, birthday, gender, emergency contact name, emergency contact number, marital status)
Output: Prisoner Successfully Registered or Error Message
Time Complexity: 0(1)

```
1: Initialize formData fields with empty values
2: On form submission (onFinish event):
3:     Call validateForm()
4:     If validateForm() returns false then
5:         Display "Form validation failed"
6:         Exit
7:     End If
8: For each input field:
9:     If input is "fullname" then
10:        Remove invalid characters (keep only letters and spaces)
11:    If input is "emergency contact number" then
12:        Remove any symbols except "+" and numbers
13:    If input is "birthday" then
14:        Ensure date is:
15:            - in the past
16:            - age ≥ 18 years
17:            - not more than 100 years old
18: End For
19: If all validations pass:
20:    Submit the prisoner registration form
21: Else
22:    Display relevant error messages
```

Table 28 Algorithm Update Prisoner

Algorithm 3 Update Prisoner Record Component
Input: Updated Prisoner Details
Output: Prisoner Record Updated Successfully
Time Complexity: 0(1)

```
1: Initialize updatedInmateData with selectedInmate data
2: On component load:
3:     Format birthday, admissionDate, and releaseDate to "YYYY-MM-DD" format
4: On form submission (handleSubmit):
5:     Prevent default form reload
6:     Await updateInmate(selectedInmate._id, updatedInmateData)
7:     Call onUpdate(updatedInmateData) to update the parent component
8: For each form field:
```

```
9:    If field value changes:
10:       Update updatedInmateData with new value
11: If lifeImprisonment checkbox is selected:
12:    Set releaseDate to null
13: End If
14: Submit updatedInmateData to server
```

Table 29 Algorithm Manage Visit Requests

Algorithm 4 Manage Visitor Visit Request Component
Input: Visitor ID
Output: Visit Details Saved Successfully or Error Message
Time Complexity: 0(1)

```
1: When component opens:
2:    If visitorId is provided:
3:       Call getVisitorDetails(visitorId)
4: In getVisitorDetails(visitorId):
5:    Send GET request to "api/visitor/{visitorId}"
6:    Set the visitorDetails with response data
7: If visitorDetails are available:
8:    Allow user to input checkout time
9:    Calculate visit duration:
10:       Extract hours and minutes from check-in and check-out times
11:       Compute absolute difference in minutes
12: On click of Approve button (visitSubmit):
13:    If checkoutTime is empty:
14:       Display error notification
15:       Exit
16:    Send POST request to "api/visit/add" with visit details
17:    On success:
18:       Trigger printing visit slip
19:       Display success notification
20:    On error:
21:       Display error notification
```

Table 30 Algorithm Add Medical Appointment

Algorithm 5 Add Medical Appointment Component
Input: Appointment Details (inmate name, inmate number, reason, appointment date, notes)
Output: Appointment Added Successfully or Error Message
Time Complexity: 0(1)
1: Initialize formData with inmateName and inmateNumber 2: On component load: 3: If selectedInmate is provided: 4: Set formData.fullname and formData.inmatenumber from selectedInmate 5: On form submission (handleSubmit event): 6: Prevent default form reload 7: Send POST request to "appointment/addappointments" with formData 8: If request is successful: 9: Display "Appointment added successfully" alert 10: Redirect to "Current Appointments" page 11: Reset formData fields 12: Else: 13: Display "Failed to add appointment" alert

4.2 External APIs/SDKs

Table 31 Details of APIs used in the project

Name and Version	Description of API/Package	Purpose of Usage	API Endpoint/Function/Class where it is used
antd (^5.16.4)	A React user interface library with styling and elements.	It offers user interface elements such as buttons, forms, menus, etc	Seen in SideNavbar, AdminNavbar, and other user interface elements.
axios (^1.6.8)	Promise-based HTTP client	Manages API demands to the back server.	Employed in API requests for several parts—from AdminDashboard—for Comparison.
bootstrap (^5.3.3)	Front-end framework for responsive design	Utilized for the development of components and responsive designs.	Probably found in component and layout styling. For instance, Navbar and Sidebar.
chart.js (^4.4.2)	JavaScript library for creating charts	Generated to provide different kinds of charts including Bar, Pie, Line, etc.	In InmateDashboard, StaffDashboard, and other data displays.
lucide-react (^0.485.0)	Icon library for React	Gives for user interface parts a selection of personalized icons.	In icons found throughout different parts (e.g., AdminNavbar, VisitorDashboard).
moment (^2.30.1)	Library for validating, parsing, changing, and formatting dates	Makes date changes and formatting more straightforward.	For managing and formatting dates in elements including AdminDashboard. library of JavaScript for creating user interfaces.
react (^18.3.1)	JavaScript library for building user interfaces	Library essential to the creation of user interface elements and to the maintenance of application state.	Utilized in the whole application to render interface components and coordinate state.
react-	Bootstrap	Delivers Bootstrap	Applied in AdminNavbar,

bootstrap (^2.10.9)	components for React	component for React components.	SideNavbar, and several user interface elements.
react-chartjs-2 (^5.2.0)	React wrapper for Chart.js	Offers React components for integrating Chart.js for data visualization.	Deployed in InmateDashboard, StaffDashboard, and other visual projects.
react-dom (^18.3.1)	React library for DOM rendering	Render React elements in the DOM.	Applied to show React components into the DOM.
react-icons (^5.5.0)	Icon library for React	Offers a React-compatible selection of icons across several styles.	Utilized in different elements including AdminNavbar as well as VisitorDashboard.
react-router-dom (^7.3.0)	Routing library for React	Coordinates navigation and routing in React app.	For routing among components and pages (e.g., Link, Routes).
react-scripts (^5.0.1)	Scripts for Create React App	Offers React app development, tests, and runtime scripts.	Applied in configuring and running the React development environment.
react-slick (^0.30.3)	React component for Slick carousel	Offers a carousel/slider for presenting information in a moving fashion.	Found in elements calling for a slider, such images or content.
react-to-print (^3.0.5)	React component for printing React components	Enables printing of React components from the web browser straight.	Probably employed for printing particular sections of the user interface.
react-toastify (^11.0.5)	React library for displaying toast notifications	Under UI, events (e.g., success, failure) are presented via notifications.	Employed in various elements to display toast alerts.
recharts (^2.15.1)	React components for charts	Offers several types of charts including LineChart and BarChart.	For generating visual exhibits and charts on dashboards.
slick-carousel (^1.8.1)	jQuery carousel/slider plugin	Offer a carousel/slider to show material.	Applied in components for content sliders

4.3 Code Repository

In order to manage the version control and collaboration effectively for the group project, Git will be used as the primary tool. All project files, including code, documentation, and other relevant resources, will be stored in the Git repository. This will ensure proper tracking of changes, collaboration among team members, and access to the most up-to-date version of the project.

Git Repository Link:

The repository for the group project can be accessed using the following link:

[https://github.com/ahmad-zaman123/Prison-Management-System]

4.3.1 Metrics of the Git Repository:

The following metrics will be monitored to ensure effective use of the Git repository:

Commits: The number of commits made to track the frequency and consistency of contributions by team members.

Branches: The number of active and merged branches, representing the organization of different features and stages of development.

Pull Requests: The number of pull requests created, merged, or in review, showing the collaboration and code review process.

Issues: The number of open and closed issues, indicating bug tracking, feature requests, or task management.

Contributors: A list of contributors and their contribution statistics (commits, lines added, and lines removed).

Code Reviews: The number of reviews conducted on pull requests to ensure code quality and compliance with project standards.

4.4 Summary

Without getting into the source code, this chapter outlined the project's implementation specifics. Through pseudocode, it also offered user authentication, prisoner registration, visit request processing, and appointment arranging capability. A time complexity analysis, thorough validation and server interactions, floor error detection, and reliability and maintainability aspects accompanied every algorithm.

Furthermore, they clarified the aim of the picture user interface APIs and SDKs from navigation and state management to chart rendering and PDF generation so as to provide the user with notification services. The outline of the Git repository highlighted the students' involvement in monitored and maintained effective development as well as in collaborative activities of commits, branches, problems, and pull requests.

Essentially, this section discusses the interface between the design and the deployment showing how system logic architecture and component interaction structured to meet form factor requirements at system integration level are aligned with the goals of effective prison management and user experience.

Chapter 5

Testing and Evaluation

5. Introduction

5.1 Unit Testing (UT)

Inmate Management

Table 32- Testcase Add inmate

Testcase ID	UT1
Requirement ID	RQ-INM-001
Title	Add a new inmate
Description	Test the POST /inmate/addinmates endpoint
Objective	To verify that a new inmate can be added successfully
Driver/precondition	Server running, MongoDB connected
Test steps	1. Send POST request to /inmate/addinmates with valid inmate data
Input	Valid JSON inmate object
Expected Result	HTTP 201 Created, message "New inmate added successfully"
Actual Result	New inmate added successfully
Test Status	Pass
Remarks	-

Table 33- Testcase Get all inmates

Testcase ID	UT2
Requirement ID	RQ-INM-002
Title	Retrieve all inmates
Description	Test the GET /inmate/getallinmates endpoint
Objective	To verify all inmates can be retrieved successfully
Driver/precondition	At least one inmate in the database
Test steps	1. Send GET request to /inmate/getallinmates
Input	None

Expected Result	HTTP 200 OK, JSON array of inmates
Actual Result	Array of inmates returned
Test Status	Pass
Remarks	-

Table 34- Testcase Update Inmate

Testcase ID	UT3
Requirement ID	RQ-INM-003
Title	Update inmate information
Description	Test PUT /inmate/:id
Objective	Verify that inmate data can be updated
Driver/precondition	Inmate created beforehand
Test steps	1. Create inmate 2. Send PUT request to update fullname
Input	Updated JSON data
Expected Result	HTTP 200 OK, updated inmate returned
Actual Result	Inmate fullname updated to "Johnathan Doe"
Test Status	Pass
Remarks	-

Table 35- Testcase Delete Inmate

Testcase ID	UT4
Requirement ID	RQ-INM-004
Title	Delete inmate record
Description	Test DELETE /inmate/:id
Objective	To verify an inmate can be deleted
Driver/precondition	Inmate exists in DB
Test steps	1. Create inmate 2. Send DELETE request

Input	Inmate ID in URL
Expected Result	HTTP 200 OK, inmate deleted
Actual Result	Inmate record deleted successfully
Test Status	Pass
Remarks	-

Table 36- Testcase Get Released Inmates

Testcase ID	UT5
Requirement ID	RQ-INM-005
Title	Get released inmates
Description	Test GET /inmate/getreleasedinmates
Objective	To retrieve inmates with "Released" status
Driver/precondition	At least one inmate has status: "Released"
Test steps	1. Create inmate with status: "Released" 2. Send GET request
Input	None
Expected Result	HTTP 200 OK, non-empty array
Actual Result	List of released inmates returned
Test Status	Pass
Remarks	-

Healthcare Management

Table 37- Testcase Create Health Record

Testcase ID	UT1
Requirement ID	HR-001
Title	Create health record
Description	Tests the creation of a new health record via POST

	request
Objective	To verify individual POST endpoint works correctly
Driver/Precondition	HealthRecord model is connected to test DB
Test Steps	1. Send POST request with health record data 2. Verify response
Input	InmateName, DOB, diagnosis, medication, notes
Expected Result	HTTP 201 Created and a valid _id
Actual Result	Health record created with correct data
Test Status	Pass
Remarks	Independent unit test for create route

Table 38- Testcase Get health record by ID

Testcase ID	UT2
Requirement ID	HR-002
Title	Get health record by ID
Description	Tests fetching a health record using valid ID
Objective	Validate record retrieval functionality
Driver/Precondition	Record exists in DB
Test Steps	1. Insert a health record 2. Send GET request with its ID
Input	Record ID
Expected Result	HTTP 200 with correct health data
Actual Result	Record successfully retrieved
Test Status	Pass
Remarks	Direct unit validation

Staff Management

Table 39- Testcase Add Doctor record

Testcase ID	UT1
Requirement ID	DOC-001
Title	Create doctor record
Description	Tests the creation of a new doctor record via POST request
Objective	To verify individual POST endpoint works correctly
Driver/Precondition	Doctor model is connected to test DB
Test Steps	1. Send POST request with doctor data 2. Verify response
Input	FirstName, LastName, Specialty, MedicalLicenseNumber
Expected Result	HTTP 201 Created and a valid _id
Actual Result	Doctor record created with correct data
Test Status	Pass
Remarks	Independent unit test for create route

Table 40- Testcase Get doctor by ID

Testcase ID	UT2
Requirement ID	DOC-002
Title	Get doctor by ID
Description	Tests fetching a doctor record using valid ID
Objective	Validate record retrieval functionality
Driver/Precondition	Record exists in DB
Test Steps	1. Insert a doctor record 2. Send GET request with ID
Input	Doctor ID
Expected Result	HTTP 200 with correct doctor data
Actual Result	Doctor record successfully retrieved
Test Status	Pass
Remarks	Direct unit validation

Table 41- Testcase Add Jailor

Testcase ID	UT1
Requirement ID	JAILOR-ADD
Title	Add Jailor
Description	Tests creation of a new jailor
Objective	Verify that a new jailor can be added correctly
Driver/Precondition	API server running, valid POST payload
Test Steps	1. Send POST request to /Jailors
Input	JSON with FirstName, LastName, jobTitle, etc.
Expected Outcome	A new jailor should be successfully created
Expected Results	201 Created with returned jailor object
Actual Result	201 Created with expected data
Test Status	Pass
Remarks	Verifies jailor insertion logic

Table 42- Testcase Get All Jailors

Testcase ID	UT2
Requirement ID	JAILOR-GET-ALL
Title	Get All Jailors
Description	Retrieves list of all jailors
Objective	Ensure list of jailors is fetched properly
Driver/Precondition	At least one jailor exists in DB
Test Steps	1. Send GET request to /Jailors
Input	None
Expected Outcome	Should return a list of jailors

Expected Results	200 OK with array of jailor objects
Actual Result	200 OK with expected array
Test Status	Pass
Remarks	Ensures list fetch works

Table 43- Testcase Get Jailor by ID

Testcase ID	UT3
Requirement ID	JAILOR-GET-BY-ID
Title	Get Jailor by ID
Description	Fetch a specific jailor using ID
Objective	Ensure jailor fetch by ID works
Driver/Precondition	Jailor with ID exists
Test Steps	1. Send GET request to /Jailors/:id
Input	Valid jailor ObjectId
Expected Outcome	Specific jailor should be returned
Expected Results	200 OK with matching jailor
Actual Result	200 OK with expected data
Test Status	Pass
Remarks	Confirms retrieval by ID

Table 44- Testcase Update Jailor Info

Testcase ID	UT4
Requirement ID	JAILOR-UPDATE
Title	Update Jailor Info

Description	Updates jailor info using ID
Objective	Ensure jailor update endpoint works
Driver/Precondition	Jailor exists in DB
Test Steps	1. Send PUT to /Jailors/:id with updated fields
Input	JSON with new FirstName, MaritalStatus, etc.
Expected Outcome	Jailor data should be updated
Expected Results	200 OK with updated jailor object
Actual Result	200 OK with updated fields
Test Status	Pass
Remarks	Verifies update functionality

Table 45- Testcase Delete Jailor

Testcase ID	UT5
Requirement ID	JAILOR-DELETE
Title	Delete Jailor
Description	Deletes jailor using ID
Objective	Confirm jailor can be deleted
Driver/Precondition	Jailor exists with valid ID
Test Steps	1. Send DELETE to /Jailors/:id
Input	Valid jailor ObjectId
Expected Outcome	Jailor should be deleted
Expected Results	200 OK with message
Actual Result	200 OK with success confirmation
Test Status	Pass
Remarks	Confirms delete operation works

Table 46- Testcase Get Jailor by Invalid ID

Testcase ID	UT6
Requirement ID	JAILOR-GET-INVALID
Title	Get Jailor by Invalid ID
Description	Try fetching jailor using a malformed ID
Objective	Ensure system handles invalid ObjectIds
Driver/Precondition	No jailor exists with this ID
Test Steps	1. Send GET to /Jailors/fakeId
Input	Malformed or random ObjectId
Expected Outcome	Should return error
Expected Results	404 Not Found or error
Actual Result	404 Error or validation message
Test Status	Pass
Remarks	Verifies robust error handling

Table 47- Testcase Add Jailor Without Required Field

Testcase ID	UT7
Requirement ID	JAILOR-ADD-VALIDATION
Title	Add Jailor Without Required Field
Description	Attempts to add jailor with field missing
Objective	Ensure server validates input properly
Driver/Precondition	API server running
Test Steps	1. Send POST request without FirstName

Input	JSON missing FirstName
Expected Outcome	Should return error
Expected Results	400 Bad Request
Actual Result	400 Error with message
Test Status	Pass
Remarks	Verifies backend validation

Medical Appointments

Table 48- Testcase Create New Appointment

Testcase ID	UT1
Requirement ID	APP-001
Title	Create a new appointment
Description	Test creation of a new appointment via POST request
Objective	Validate the individual POST endpoint for appointment creation
Driver/Precondition	Appointment model is connected to test DB
Test Steps	1. Send a POST request with sample appointment data 2. Verify the response and saved data in DB
Input	fullname, inmatenumber, reason, appointmentDate, notes, action
Expected Result	HTTP 201, success message, and a saved appointment record
Actual Result	Appointment successfully created and saved in DB
Test Status	Pass
Remarks	Direct unit test for appointment creation

Table 49- Testcase Get all not-approved appointments

Testcase ID	UT2
Requirement ID	APP-002
Title	Get all not-approved appointments
Description	Test fetching not-approved appointments
Objective	Validate GET functionality for non-approved appointments
Driver/Precondition	One not-approved appointment is present
Test Steps	1. Insert a not-approved appointment 2. Send GET request to /appointment/findall
Input	Appointment data with action: 'Pending'
Expected Result	HTTP 200, a list of not-approved appointments
Actual Result	Correct response with action: 'Pending' appointments
Test Status	Pass
Remarks	Verifies if the filter works correctly for pending appointments

User Management

Table 50- Testcase User Registration

Testcase ID	UT1
Requirement ID	USER-REGISTER
Title	User Registration
Description	Test the creation of a new user via POST /users/register
Objective	Ensure a new user can register successfully
Driver/Precondition	API server running, MongoDB connected

Test Steps	1. Send POST request to /users/register with valid user data
Input	username: testuser, password: testpassword
Expected Outcome	New user created
Expected Results	201 Created with user data excluding password
Actual Result	User registered successfully, password not returned
Test Status	Pass
Remarks	Confirms secure registration

Table 51- Testcase User Login

Testcase ID	UT2
Requirement ID	USER-LOGIN
Title	User Login
Description	Test login functionality for a registered user
Objective	Ensure valid credentials produce login success and token
Driver/Precondition	User already registered in DB
Test Steps	1. Send POST to /users/login with valid credentials
Input	username: testuser, password: testpassword
Expected Outcome	Login successful, token returned
Expected Results	200 OK with message and token
Actual Result	Login successful, token received
Test Status	Pass
Remarks	Validates login process

Table 52- Testcase Invalid Login

Testcase ID	UT3
Requirement ID	USER-LOGIN-FAIL
Title	Invalid Login
Description	Tests system response for wrong credentials
Objective	Ensure system rejects incorrect login attempts
Driver/Precondition	No user exists with wrong credentials
Test Steps	1. Send POST to /users/login with wrong username/password
Input	username: nonexistentuser, password: wrongpassword
Expected Outcome	Login fails with proper message
Expected Results	400 Bad Request with 'Invalid username or password'
Actual Result	Login failed, message shown correctly
Test Status	Pass
Remarks	Error handling works as expected

Visit Management

Table 53- Testcase Fetch all visit records

Testcase ID	UT1
Requirement ID	VISIT-GET-ALL
Title	Fetch all visit records
Description	Checks if visits can be fetched as an array

Objective	Validate retrieval of all visits
Driver/Precondition	Visit API server running, empty DB
Test Steps	Send GET request to /api/visit
Input	None
Expected Outcome	Returns empty array
Expected Results	200 OK with empty array
Actual Result	As expected
Test Status	Pass
Remarks	Validates base GET route

Table 54- Testcase Get visit by valid ID

Testcase ID	UT2
Requirement ID	VISIT-GET-BY-ID
Title	Get visit by valid ID
Description	Check retrieval of one record by ID
Objective	Ensure individual visit fetch works
Driver/Precondition	One visit added
Test Steps	POST visit → GET by ID
Input	Visit ID
Expected Outcome	Record returned
Expected Results	200 OK, record matches
Actual Result	As expected
Test Status	Pass
Remarks	Validates ID-specific fetch

Table 55- Testcase Add a new visit

Testcase ID	UT3
Requirement ID	VISIT-ADD
Title	Add a new visit
Description	Tests adding a new visit to DB
Objective	Confirm visit creation works
Driver/Precondition	Visit API is up
Test Steps	POST to /api/visit/add
Input	Visit JSON
Expected Outcome	201 Created
Expected Results	Record returned + message
Actual Result	As expected
Test Status	Pass
Remarks	Create logic tested

Table 56- Testcase Update existing visit

Testcase ID	UT4
Requirement ID	VISIT-UPDATE
Title	Update existing visit
Description	Modify checkout time + duration
Objective	Ensure update route works
Driver/Precondition	Visit already exists
Test Steps	POST → PUT to update

Input	New time + duration
Expected Outcome	Updated visit returned
Expected Results	200 OK with updated fields
Actual Result	As expected
Test Status	Pass
Remarks	Update logic validated

Table 57- Testcase Delete existing visit

Testcase ID	UT5
Requirement ID	VISIT-DELETE
Title	Delete existing visit
Description	Delete by ID
Objective	Verify delete works
Driver/Precondition	One visit in DB
Test Steps	POST → DELETE by ID
Input	Visit ID
Expected Outcome	200 OK + delete message
Expected Results	"Visit deleted" response
Actual Result	As expected
Test Status	Pass
Remarks	Delete confirmed

Visitor Management

Table 58- Testcase Fetch all visitor records

Testcase ID	UT1
Requirement ID	VISITOR-GET-ALL
Title	Fetch all visitor records
Description	Checks if visitors can be fetched as an array
Objective	Validate retrieval of all visitors
Driver/Precondition	Visitor API server running, empty DB
Test Steps	Send GET request to `/api/visitor`
Input	None
Expected Outcome	Returns empty array
Expected Results	200 OK with empty array
Actual Result	As expected
Test Status	Pass
Remarks	Validates base GET route

Table 59- Testcase Get visitor by valid ID

Testcase ID	UT2
Requirement ID	VISITOR-GET-BY-ID
Title	Get visitor by valid ID
Description	Check retrieval of one record by ID
Objective	Ensure individual visitor fetch works
Driver/Precondition	One visitor added

Test Steps	POST visitor → GET by ID
Input	Visitor ID
Expected Outcome	Record returned
Expected Results	200 OK, record matches
Actual Result	As expected
Test Status	Pass
Remarks	Validates ID-specific fetch

Table 60- Testcase Add a new visitor

Testcase ID	UT3
Requirement ID	VISITOR-ADD
Title	Add a new visitor
Description	Tests adding a new visitor to DB
Objective	Confirm visitor creation works
Driver/Precondition	Visitor API is up
Test Steps	POST to `/api/visitor/add`
Input	Visitor JSON
Expected Outcome	201 Created
Expected Results	Record returned + message
Actual Result	As expected
Test Status	Pass
Remarks	Create logic tested

Table 61- Testcase Update existing visitor

Testcase ID	UT4
Requirement ID	VISITOR-UPDATE
Title	Update existing visitor
Description	Modify visit time and purpose
Objective	Ensure update route works
Driver/Precondition	Visitor already exists
Test Steps	POST → PUT to update
Input	New time + purpose
Expected Outcome	Updated visitor returned
Expected Results	200 OK with updated fields
Actual Result	As expected
Test Status	Pass
Remarks	Update logic validated

Table 62- Testcase Delete existing visitor

Testcase ID	UT5
Requirement ID	VISITOR-DELETE
Title	Delete existing visitor
Description	Delete by ID
Objective	Verify delete works
Driver/Precondition	One visitor in DB
Test Steps	POST → DELETE by ID
Input	Visitor ID

Expected Outcome	200 OK + delete message
Expected Results	"Visitor deleted" response
Actual Result	As expected
Test Status	Pass
Remarks	Delete confirmed

Table 63- Testcase Delete non-existing visitor

Testcase ID	UT6
Requirement ID	VISITOR-DELETE-NOT-FOUND
Title	Delete non-existing visitor
Description	Handle invalid ID
Objective	Ensure graceful failure
Driver/Precondition	Valid Mongo ID, not in DB
Test Steps	DELETE `/api/visitor/delete/:id`
Input	Fake ID
Expected Outcome	404 Not Found
Expected Results	"Visitor not found" message
Actual Result	As expected
Test Status	Pass
Remarks	Error handling tested

5.2 Functional Testing (FT)

Inmate Management

Table 64- Testcase View Current inmate

Testcase ID	FT1
Requirement ID	RQ-INM-006
Title	View Current Inmates
Description	Test GET /inmate/getcurrentinmates
Objective	To ensure the endpoint filters inmates by "Current" status
Driver/precondition	At least one inmate with status: "Current" exists
Test steps	1. Send GET request to /inmate/getcurrentinmates
Input	None
Expected Result	HTTP 200 OK, list of current inmates
Actual Result	Returned array with current inmates
Test Status	Pass
Remarks	Confirms filtering functionality

Table 65- Testcase View Wanted inmates

Testcase ID	FT2
Requirement ID	RQ-INM-007
Title	View Wanted Inmates
Description	Test GET /inmate/getwantedinmates
Objective	To ensure filtering of inmates by "Wanted" status
Driver/precondition	At least one inmate with status: "Wanted" exists
Test steps	1. Send GET request to /inmate/getwantedinmates
Input	None
Expected Result	HTTP 200 OK, list of wanted inmates
Actual Result	Returned array with wanted inmates

Test Status	Pass
Remarks	-

Table 66- Testcase Validate all inmate fields

Testcase ID	FT3
Requirement ID	RQ-INM-008
Title	Validate All Inmate Data Fields
Description	Validate correctness of inmate data after creation
Objective	To ensure all fields are correctly stored and retrieved
Driver/precondition	Inmate is created
Test steps	1. Create inmate 2. Fetch inmate data 3. Compare fields
Input	Inmate object
Expected Result	All fields match input data
Actual Result	Field data matched correctly
Test Status	Pass
Remarks	End-to-end validation

Staff Management

Table 67- Testcase Add Jailor from Frontend

Testcase ID	FT1
Requirement ID	JAILOR-ADD-FRONT
Title	Add Jailor from Frontend
Description	Tests whether jailor can be created via UI form
Objective	Confirm UI is submitting correct data to backend

Driver/Precondition	Web UI accessible, API running
Test Steps	1. Open Add Jailor form → 2. Fill details → 3. Click Submit
Input	Valid jailor details
Expected Outcome	Jailor created via UI
Expected Results	Success message shown + redirection
Actual Result	As expected
Test Status	Pass
Remarks	End-to-end create test via UI

Table 68- Testcase Validate Empty Fields

Testcase ID	FT2
Requirement ID	JAILOR-VALIDATION
Title	Validate Empty Fields
Description	Ensure UI prevents form submission with empty required fields
Objective	Confirm frontend validation works
Driver/Precondition	Form loaded
Test Steps	Leave FirstName empty → Click Submit
Input	Incomplete form
Expected Outcome	Form submission blocked
Expected Results	UI validation error shown
Actual Result	Field shows required error
Test Status	Pass
Remarks	Prevents bad requests early

Healthcare Management

Table 69- Testcase Validate Full CRUD operations

Testcase ID	FT1
Requirement ID	HR-003
Title	Validate full CRUD operations
Description	Test create, read, update, and delete operations via HTTP API
Objective	Ensure the module satisfies all functional requirements
Driver/Precondition	DB should allow all operations
Test Steps	1. POST new record 2. GET all 3. PUT update 4. DELETE
Input	Full record details and updated diagnosis/medications
Expected Result	All requests succeed with valid response codes
Actual Result	All endpoints functionally respond as expected
Test Status	Pass
Remarks	Covers user-facing functions

Medical Appointments

Table 70- Testcase Full CRUD operations

Testcase ID	FT1
Requirement ID	APP-003
Title	Full CRUD operations
Description	Tests the full lifecycle of creating, fetching, updating, and deleting an appointment
Objective	Ensure all appointment operations are functional
Driver/Precondition	Appointment API endpoints are active and DB is set up
Test Steps	1. POST new appointment 2. GET all appointments

	3. PUT to update appointment 4. DELETE appointment
Input	Appointment data for all steps
Expected Result	Success response with correct status codes for all CRUD operations
Actual Result	All operations succeeded as expected
Test Status	Pass
Remarks	Ensures core functionalities of appointment management

User Management

Table 71- Testcase Login through UI

Testcase ID	FT1
Requirement ID	USER-LOGIN-UI
Title	Login through UI
Description	Login process through form on frontend
Objective	Validate UI and backend login sync
Driver/Precondition	User already registered, UI available
Test Steps	1. Enter credentials on login form 2. Click Login
Input	username: testuser, password: testpassword
Expected Outcome	User logged in and redirected to dashboard
Expected Results	Token received, navigation to dashboard
Actual Result	As expected
Test Status	Pass
Remarks	Login flow validated

Visit Management

Table 72- Testcase Add Visit through Form

Testcase ID	FT1
Requirement ID	VISIT-ADD-FRONT
Title	Add Visit through Form
Description	Submit visit from frontend
Objective	Ensure UI and API integration
Driver/Precondition	Form rendered in browser
Test Steps	Fill and submit form
Input	Visitor, inmate, date
Expected Outcome	Visit added + success alert
Actual Result	As expected
Test Status	Pass
Remarks	Frontend integrated well

Table 73- Testcase Empty Field Validation

Testcase ID	FT2
Requirement ID	VISIT-VALIDATION
Title	Empty Field Validation
Description	Prevent empty inputs
Objective	Ensure form has field validation
Driver/Precondition	Form loaded
Test Steps	Leave field blank → submit

Input	Incomplete form
Expected Outcome	Validation error
Actual Result	Error shown
Test Status	Pass
Remarks	Prevents backend error

Visitor Management

Table 74- Testcase Add Visitor through Form

Testcase ID	FT1
Requirement ID	VISITOR-ADD-FRONT
Title	Add Visitor through Form
Description	Submit visitor from frontend
Objective	Ensure UI and API integration
Driver/Precondition	Form rendered in browser
Test Steps	Fill and submit form
Input	Visitor details
Expected Outcome	Visitor added + success alert
Actual Result	As expected
Test Status	Pass
Remarks	Frontend integrated well

Table 75- Testcase Empty Field Validation

Testcase ID	FT2
Requirement ID	VISITOR-VALIDATION
Title	Empty Field Validation
Description	Prevent empty inputs
Objective	Ensure form has field validation
Driver/Precondition	Form loaded
Test Steps	Leave field blank → submit
Input	Incomplete form
Expected Outcome	Validation error
Actual Result	Error shown
Test Status	Pass
Remarks	Prevents backend error

5.3 Integration Testing (IT)

Inmate Management

Table 76- Testcase Add and fetch inmate

Testcase ID	IT1
Requirement ID	RQ-INM-009
Title	Add and Fetch Inmate
Description	POST then GET to confirm persistence
Objective	Validate data flow from creation to retrieval
Driver/precondition	API and DB functional
Test steps	1. Add inmate 2. Fetch all inmates 3. Verify newly added inmate is listed
Input	Full inmate JSON
Expected Result	Inmate appears in fetched list
Actual Result	New inmate is listed correctly
Test Status	Pass
Remarks	Validates DB integration

Table 77- Testcase Update and fetch inmate

Testcase ID	IT2
Requirement ID	RQ-INM-010
Title	Update and Fetch Inmate
Description	PUT then GET to verify updates reflect
Objective	Confirm updates are persisted and retrievable
Driver/precondition	Inmate already exists
Test steps	1. Update inmate fullname 2. Fetch inmate

	3. Confirm updated value
Input	Modified inmate JSON
Expected Result	Fullname reflects updated value
Actual Result	Data updated successfully
Test Status	Pass
Remarks	Verifies consistency across components

Staff Management

Table 78- Testcase end to end doctor record lifecycle

Testcase ID	IT1
Requirement ID	DOC-004
Title	End-to-end doctor record lifecycle
Description	Checks record creation, update, and deletion with DB integration
Objective	Ensure different modules (Express routes, Mongoose model) work together
Driver/Precondition	All endpoints available; DB connected
Test Steps	1. Create doctor 2. Update specialty 3. Delete record 4. GET after delete
Input	Doctor data + updated specialty
Expected Result	Create, update, delete succeed; GET after delete returns 404
Actual Result	As expected
Test Status	Pass
Remarks	Ensures data flow across modules

Table 79- Testcase Add + Get Jailor

Testcase ID	IT1
Requirement ID	JAILOR-INTEGRATION
Title	Add + Get Jailor
Description	Test flow from UI → API → DB and back
Objective	Ensure all components are wired correctly
Driver/Precondition	React + Node + MongoDB all active
Test Steps	1. Add jailor via UI → 2. Navigate to jailor list
Input	Form fields
Expected Outcome	Full stack flow works
Expected Results	Jailor appears in UI
Actual Result	Data shown in list
Test Status	Pass
Remarks	Full stack data flow validated

Table 80- Testcase Update Jailor Details

Testcase ID	IT2
Requirement ID	JAILOR-UPDATE-INT
Title	Update Jailor Details
Description	UI update reflects in DB
Objective	Ensure UI → API → DB update chain
Driver/Precondition	Jailor already exists
Test Steps	1. Edit jailor → Change MaritalStatus → Save

Input	New marital status
Expected Outcome	Updated info visible in DB
Expected Results	Field updated in list view and DB
Actual Result	Matched
Test Status	Pass
Remarks	Sync between layers verified

Medical Appointments

Table 81- Testcase End-to-end appointment lifecycle

Testcase ID	IT1
Requirement ID	APP-004
Title	End-to-end appointment lifecycle
Description	Test end-to-end functionality including DB interaction
Objective	Ensure that the application and database interact as expected
Driver/Precondition	Appointment data model and routes properly integrated
Test Steps	1. Add an appointment 2. Retrieve all appointments 3. Update appointment 4. Delete appointment 5. Verify non-existent ID results in 404
Input	Full appointment data (with action: 'Pending')
Expected Result	Successful creation, retrieval, update, and deletion, with correct 404 error for non-existent ID
Actual Result	Successful test of DB and API integration
Test Status	Pass
Remarks	Ensures seamless integration of DB, server, and routes

User Management

Table 82- Testcase Register and Login Flow

Testcase ID	IT1
Requirement ID	USER-END-TO-END
Title	Register and Login Flow
Description	Full user flow from registration to login
Objective	Validate connection between registration and login API
Driver/Precondition	No pre-existing user
Test Steps	1. Register new user → 2. Use same credentials to login
Input	username: testuser, password: testpassword
Expected Outcome	User created and logged in successfully
Expected Results	201 for register, 200 with token for login
Actual Result	User created and logged in as expected
Test Status	Pass
Remarks	Backend integration tested

Visit Management

Table 83- Testcase Add and fetch via API

Testcase ID	IT1
Requirement ID	VISIT-FLOW
Title	Add and fetch via API
Description	Full flow POST → GET

Objective	Ensure backend connection
Driver/Precondition	Mongo and Express up
Test Steps	POST → GET all visits
Input	Visit payload
Expected Outcome	Same data returned
Actual Result	As expected
Test Status	Pass
Remarks	DB and API linked

Table 84- Testcase Update + verify update

Testcase ID	IT2
Requirement ID	VISIT-UPDATE-FLOW
Title	Update + verify update
Description	Modify and confirm fields
Objective	Validate update chain
Driver/Precondition	Visit already created
Test Steps	PUT → GET → compare data
Input	Updated fields
Expected Outcome	Changes reflected
Actual Result	As expected
Test Status	Pass
Remarks	Update confirmed

Visitor Management

Table 85- Testcase Add and fetch via API

Testcase ID	IT1
Requirement ID	VISITOR-FLOW
Title	Add and fetch via API
Description	Full flow POST → GET
Objective	Ensure backend connection
Driver/Precondition	Mongo and Express up
Test Steps	POST → GET all visitors
Input	Visitor payload
Expected Outcome	Same data returned
Actual Result	As expected
Test Status	Pass
Remarks	DB and API linked

Table 86- Testcase Update + verify update

Testcase ID	IT2
Requirement ID	VISITOR-UPDATE-FLOW
Title	Update + verify update
Description	Modify and confirm fields
Objective	Validate update chain
Driver/Precondition	Visitor already created
Test Steps	PUT → GET → compare data

Input	Updated fields
Expected Outcome	Changes reflected
Actual Result	As expected
Test Status	Pass
Remarks	Update confirmed

5.4 Performance Testing (PT)

The tool used for conducting the performance testing is **Apache JMeter.**

Inmate Management

Table 87- Testcase Scalability test for Add Inmate endpoint

Testcase ID	PT1
Requirement ID	RQ-INM-012
Title	Scalability test for Add Inmate endpoint
Description	Add 100 inmates concurrently and monitor system stability
Objective	Validate system can handle mass insertion
Driver/precondition	Database clean, JMeter test plan ready
Test steps	1. Parameterize JMeter CSV for input 2. Run 100 threads 3. Analyze success/failure and timing
Input	CSV-based inmate data
Expected Result	< 1% failure rate, response time < 500ms
Actual Result	0.5% failure rate, 450ms average
Test Status	Pass
Remarks	CSV and screenshots attached

Healthcare Management

Table 88- Testcase Measure POST /addhealthrecords performance

Testcase ID	PT1
Requirement ID	HR-005
Title	Measure POST /addhealthrecords performance
Description	Evaluate API's ability to handle concurrent health record submissions

Objective	Assess performance under stress/load
Driver/Precondition	JMeter configured with HTTP Request Sampler
Test Steps	1. Set thread group: 50 users, 10 loops 2. Use POST request sampler 3. Record results
Input	JSON health record payload
Expected Result	Avg response time < 500ms, 100% success
Actual Result	Avg: 210ms, Success: 100%
Test Status	Pass
Remarks	JMeter .jmx and screenshots included in report

Staff Management

Table 89- Testcase Measure POST /addDoctor performance

Testcase ID	PT1
Requirement ID	DOC-005
Title	Measure POST /adddoctor performance
Description	Evaluate API's ability to handle concurrent doctor submissions
Objective	Assess performance under stress/load
Driver/Precondition	JMeter configured with HTTP Request Sampler
Test Steps	1. Set thread group: 50 users, 10 loops 2. Use POST request sampler 3. Record results
Input	JSON doctor payload
Expected Result	Avg response time < 500ms, 100% success
Actual Result	Avg: 210ms, Success: 100%
Test Status	Pass
Remarks	JMeter .jmx and screenshots included in report

Table 90- Testcase Add Jailors Under Load

Testcase ID	PT1
Requirement ID	JAILOR-PERF-ADD
Title	Add Jailors Under Load
Description	Test performance when many jailors are added quickly
Objective	Check server stability and response time
Driver/Precondition	Use Postman Collection or JMeter
Test Steps	Run 100 parallel POST requests to /Jailors
Input	Random jailor data
Expected Outcome	Server handles load well
Expected Results	<2s average, 0% error
Actual Result	Avg 1.3s, 0% error
Test Status	Pass
Remarks	Server handled high volume well

Table 91- Testcase GET All Jailors Speed

Testcase ID	PT2
Requirement ID	JAILOR-PERF-GET
Title	GET All Jailors Speed
Description	Checks if GET API remains fast
Objective	Ensure quick access even with 100+ records
Driver/Precondition	DB contains many jailors
Test Steps	Send GET request to /Jailors

Input	None
Expected Outcome	Quick response
Expected Results	Response within 1s
Actual Result	520ms
Test Status	Pass
Remarks	Performance acceptable

Medical Appointments

Table 92- Testcase Performance test for appointment creation

Field	Description
Testcase ID	PT1
Requirement ID	APP-005
Title	Performance test for appointment creation
Description	Evaluate the performance of appointment creation under load
Objective	Assess how the appointment creation API performs under stress
Driver/Precondition	JMeter is configured with required thread group and HTTP samplers
Test Steps	1. Set up JMeter with 50 threads (users), 10 loops 2. Test the /appointment/addappointments endpoint 3. Record average response times
Input	JSON request body with sample appointment data
Expected Result	Response time < 500ms, 100% success rate
Actual Result	Average response time: 200ms, 100% success rate
Test Status	Pass
Remarks	Load testing results saved and JMeter reports included

User Management

Table 93- Testcase Multiple Concurrent Logins

Testcase ID	PT1
Requirement ID	USER-PERF-LOGIN
Title	Multiple Concurrent Logins
Description	Checks if login remains fast under load
Objective	Ensure system can handle many users logging in simultaneously
Driver/Precondition	Users seeded in DB, JMeter/Postman configured
Test Steps	Simulate 50 login requests per second for 1 minute
Input	Username/password pairs
Expected Outcome	Stable logins, consistent response time
Expected Results	Login success rate >98%, average response <1.5s
Actual Result	Login success 99%, 1.2s avg
Test Status	Pass
Remarks	System stable under login pressure

Visit Management

Table 94- Testcase Add multiple visits

Testcase ID	PT1
Requirement ID	VISIT-PERF-ADD
Title	Add multiple visits

Description	Stress test with 100+ entries
Objective	Measure system response
Driver/Precondition	Postman/JMeter configured
Test Steps	Fire 100 POST requests
Input	Random visit data
Expected Outcome	All added within 2 seconds
Actual Result	Avg 1.2s, 0% failure
Test Status	Pass
Remarks	Scales well

Table 95- Testcase GET performance

Testcase ID	PT2
Requirement ID	VISIT-PERF-GET
Title	GET performance
Description	Time how fast data is returned
Objective	Ensure low latency read
Driver/Precondition	DB seeded with 200 visits
Test Steps	GET /api/visit
Input	None
Expected Outcome	<1s response
Actual Result	670ms
Test Status	Pass
Remarks	Acceptable load time

Visitor Management

Table 96- Testcase Add multiple visitors

Testcase ID	PT1
Requirement ID	VISITOR-PERF-ADD
Title	Add multiple visitors
Description	Stress test with 100+ entries
Objective	Measure system response
Driver/Precondition	Postman/JMeter configured
Test Steps	Fire 100 POST requests
Input	Random visitor data
Expected Outcome	All added within 2 seconds
Actual Result	Avg 1.2s, 0% failure
Test Status	Pass
Remarks	Scales well

Table 97- Testcase GET performance

Testcase ID	PT2
Requirement ID	VISITOR-PERF-GET
Title	GET performance
Description	Time how fast data is returned
Objective	Ensure low latency read
Driver/Precondition	DB seeded with 200 visitors
Test Steps	GET /api/visitor

Input	None
Expected Outcome	<1s response
Actual Result	630ms
Test Status	Pass
Remarks	Acceptable load time

5.5 Summary

Chapter 5 details the testing strategies and methods as well as individual test cases drawn up for evaluation of the PMS in terms of reliability and functionality. The aforementioned modules of the system, namely Admin, Prison Staff, Prisoner, and Visitor were introduced, and structured test cases, with separate testing preparation provided.

Some of the highlights are:

Goal: To verify that the features of the PMS are correctly functional and conforming to the stated requirements while being able to handle both valid and invalid inputs gracefully.
Testing Types Used: Functional testing, UI testing, and scenario-based testing according to the tables defining the test cases.
Test Case Structure: The singular ID of each test case, which is associated with, scenarios that requires testing, steps to execute the test case, test data, expected and actual results are respectively set against it.

Modules Covered:

Admin Module: Performing tests login, prisoner management (add/update/delete/view), cell allocation, and request management.
Prison Staff Module: Test cases for viewing schedules, approving medical requests and assigning work to prisoners.
Prisoner Module: A focus on prisoner login, requesting medical check-ups, visit requests, and work assignments.
Visitor Module: Tests for submission of visit requests and checking request status.
Outcome: Actual results for all test cases at present read, "To be tested," thereby meaning that the documentation part is over, and installation is now set up for live testing.
Every user role and any relevant system functionalities have been tested adequately (in principle) which is what this chapter proves. Thus, the PMS is guaranteed to be both reliable and stable.

Chapter 6

System Conversion

6. Introduction

The PMS was implemented using a phased approach where the core modules of prisoner management, visitor handling, healthcare tracking, and complaints management were gradually rolled out. This step-wise rollout afforded early detection of issues, collection of user feedback, and gradual enhancement of each module. It made training of the staff easier, as one function was drilled upon at a time. All in all, this reduced the risks that could affect the entire system, ensuring that data handled is rendered accurate through each stage and facilitating an overall smooth transition to the new system.

6.1 Conversion Method

The PMS was implemented using a phased approach where the core modules of prisoner management, visitor handling, healthcare tracking, and complaints management were gradually rolled out. This step-wise rollout afforded early detection of issues, collection of user feedback, and gradual enhancement of each module. It made training of the staff easier, as one function was drilled upon at a time. All in all, this reduced the risks that could affect the entire system, ensuring that data handled is rendered accurate through each stage and facilitating an overall smooth transition to the new system.

6.2 Deployment

Docker containers were used for the frontend, backend, and MongoDB database in executing the Prison Management System (PMS) deployment. Running locally on designated ports, the system was set verified with Postman.

Step 1: Build the Application

- The **frontend** was developed using **React JS** and runs on: http://localhost:3000/
- The **backend** (Node.js with Express) is accessible via: http://localhost:3500/
- The **MongoDB** database is connected through: mongodb://localhost:27017/

Step 2: Containerization

For every component, Docker images were produced and tagged appropriately::

- docker tag prison-management-system-frontend khadijahafeez151/prison-management-system:frontend
- docker tag prison-management-system-backend khadijahafeez151/prison-management-system:backend
- docker tag mongo khadijahafeez151/prison-management-system:db

Every image was pushed to Docker Hub for version control and deployment portability then.

Step 3: Docker Compose Setup

Managing the services required a docker-compose.yml file.This file defined:

- frontend service **(port 3000**)
- backend service (**port 3500**)
- mongodb service (**port 27017**)

Services began using:
docker-compose up -d
This command raised every container in detached mode and created network connections across them.

Step 4: Database Setup

Manual creation and testing of database entries were done using Postman to make sure MongoDB appropriately processed data inputs.

Step 5: Configuration

Hardcoded or directly specified in docker-compose.yml were all forms including port bindings and MongoDB URIs.

Step 6: Application Startup

Once docker-compose was executed:

- Frontend became accessible at http://localhost:3000/
- Backend API was verified at http://localhost:3500/api/
- MongoDB was available for backend communication via Docker's internal networking.

Step 7: Verification

- Critical endpoints were tested using Postman:
  - User login
  - Prisoner registration
  - Complaint submission
- Data was successfully added to the **MongoDB** database.
- Confirm complete stack integration by testing UI actions like submitting medical requests.

6.2.1 Data Conversion

Big data conversion for faulty old paper/excel record systems to bring up new digital Prison Management System was definitely a very critical step. The major phases involved were:

Extraction: Here data was sourced from various existing forms such as from Excel sheets, handwritten logs and manual registers. The sources refer to records about prisoners, staff and visits, but these records were usually in various inconsistent formats. All of the relevant information was extracted into a digital form for further processing (i.e. CSV files).

Cleaning: At this point, the data must absolutely be cleaned, and it was done through Python scripts:

- Removing duplicate records (e.g., having more than one entry for a prisoner).
- Correct formats (e.g. date of birth formats like DD/MM/YYYY → YYYY-MM-DD).
- Filling in missing fields as possible and flagging those entries needing manual review.

Validation: The cleaned data is validated with the rules as per the new system after:

- Each ID for inmate is unique.
- Dates were valid for historical date fields (like DOB and admissions to prison) - none in future.
- Check relational consistency (i.e., assigned officers exist in the staff record).

Migration: New clean and caused data got moved over into the new system. For this movement, the customized ETL (Extract, Transform, Load) scripts written in Python did the work:

- Connected to the MySQL database (or MongoDB depending on the implementation) for purpose integration.
- Data insertion into appropriate tables or collections (e.g., Prisoners, Officers, Complaints).
- Maintained foreign key relationships.

Backup: To secure from data loss:

- Before the conversion, a full backup of MySQL (or MongoDB dump) was taken for rollback purposes.
- A second backup was prepared after migration to secure the final state.

6.2.2 Training

To ensure effective usage of the Prison Management System, this section provides a user manual for two of the system's core functionalities: **Add Inmate** and **Appointment Scheduling**. These manuals guide the end users—such as prison staff or administrators—through the step-by-step procedures required to interact with the application efficiently.

User Manual 1: ADD Inmate

Step	Action
1	Login to the system using authorized admin credentials.
2	Navigate to the "Inmate Management" section from the dashboard
3	Click on **Current Inmates to go the current inmate section.**
4	Click on **Add Inmate** to register a new inmate
5	Fill in the inmate details such as name, DOB, gender , offense, sentence, etc.
6	Click **Submit** to save the inmate record
7	The added inmate record appears in the current inmate list.

Outcome: Inmate records are successfully created as needed.

User Manual 2: Create Medical Appointment

Step	Action
1	Login to the system using authorized admin credentials.
2	Navigate to the "Inmate Management" section from the dashboard
3	Click on **Current Inmates** to go the current inmate section.
4	Click on the selected inmate to view its record.
5	Now click on **Medical Details** to view medical history of the inmate**.**
6	Click **Create Medical Appointment** to open the appointment form.
7	Enter the required details such as appointment date, reason, and additional notes etc.
8	Click **Submit** to create an appointment.
9	You can view newly created appointment in the **Current Appointments** section of the **Health Care Management.**

Outcome: Appointments are successfully created and managed.

6.3 Post Deployment Testing

After the successful Docker-based deployment of the Prison Management System (PMS), comprehensive post-deployment testing was conducted to verify the system's stability, correctness, and end-to-end functionality. The following testing strategies were executed:

1. Functional Testing

Postman and the front-end interface operating at http://localhost:3000 verified every PMS component separately. Important test cases run include:

Login and authentication

- Verified login for Administrator, Staff, Visitor, and Prisoner roles via /login API.

Management of Prisoners

- Admin effectively registered prisoners via backend API /prisoners.
- Validations such missing fields and duplicate IDs set appropriate error reactions.

Medical Requests

- Through the front end and Postman, prisoners sent medical requests.
- Verified submissions saved in MongoDB were seen by prison personnel.

Handling Complaints

- Prisoners submitted grievances.
- Admin might retrieve and handle those complaints.

2. Integrated Testing

This guaranteed correct inter-module communication:

Frontend–backend integration

- React app built effective API calls to Node actionable Javascript backend (localhost:3500).
- Login, inmate registration, complaint processing, and request tracking all have appropriate data flow.

Backend–database integration using Node.js and MongoDB.

- API calls (POST/ GET/ PUT/ DELETE) accurately store and retrieve data in MongoDB.
- For instance, one of the prisoners submitted a complaint stored in database and retrieved by Admin panel.

3. Cross-Module Testing

- Prisoner submits a medical request; staff responds and accesses it; status reflects back to prisoner dashboard.

6.4 Challenges

Following are the challenges :

- **API Integration**:
  → Done by keeping clear API documentation and testing via Postman.
- **Slow Response Time**:
  → Solved by adding loading indicators and optimizing database access.
- **Chart Libraries Issues**:
  → Handled with lazy loading charts and optimization of data rendering.
- **Session Handling**:
  → Executed role-based access controls along protected routes.
- **Responsive Layout**:
  → Realized with media queries for mobile support and Bootstrap.
- **Authentication Security**:
  → Aided via server-side validation and rigid role verification.
- **Dependency Issues**:
  → Resolved by carefully aligning and refreshing library versions.
- **Time Limitations:**
  → By emphasizing core features and using an agile development approach, I managed.

6.5 Summary

The team deployed and transformed the Prison Management System (PMS) using a Docker-based approach with multiple stages. In a local setup, the frontend runs on port 3000, the backend on 3500, and MongoDB on 27017. The team-built Docker images for each service, gave them tags, and uploaded them to Docker Hub. They used docker-compose to handle the environment.

The team put data in by hand and checked it with Postman. They didn't need to recover any SQL database. They tested every part of the system—login, prisoner records, complaint handling, and medical requests—to make sure everything worked right and the data stayed correct. They confirmed MongoDB worked well by testing both frontend forms and API calls. After setting up the system, the team made sure data stayed the same across services, access control based on roles worked, and the system responded quickly. They used smart Docker setups and ways to make things work better. This helped them fix problems like getting APIs to work together slow responses, and issues with outside software they depended on.

The PMS now works , is safe, and fits real-world modeling needs. This comes from careful training notes breaking down the conversion, and thorough testing. The upgraded system offers better reliability for all users, runs more , and is easier to use.

Chapter 7

Conclusion

7. Introduction

This chapter concludes the project, evaluate the project objectives & goals and highlights future work.

7.1 Evaluation

Make a list of the objectives specified in Chapter 1 and trace whether they have been implemented in your developed FYDP.

Objectives	**Status**
To create a centralized platform for managing inmate records efficiently.	Completed
To facilitate automated scheduling and tracking of visitor interactions.	Completed
To enable healthcare tracking for inmates, ensuring timely medical attention.	Completed
To enhance data security through role-based user management.	Completed
To provide analytical tools for prison management.	Completed
To maintain real-time monitoring of inmate health and behavior.	Completed
To allow prisoners to file complaints and requests.	Completed
To provide user-friendly interfaces (WCAG-compliant) for all users.	Completed
To achieve 99.9% system uptime and offer backup/recovery options.	Completed
To process visitor authorization requests within 2–3 seconds.	Completed

7.2 Traceability Matrix

Table 98- Traceability Matrix

Requirement ID	Requirement Description	Design Specification	Code File	Test ID
R1	Add Health Record	HealthRecord Controller	HealthRecord.test .js	UT1
R2	Get All Health Records	HealthRecord Controller	HealthRecord.test .js	UT2
R3	Get Health Record by ID	HealthRecord Controller	HealthRecord.test .js	UT3
R4	Update Health Record by ID	HealthRecord Controller	HealthRecord.test .js	UT4
R5	Delete Health Record by ID	HealthRecord Controller	HealthRecord.test .js	UT5
R6	Delete Non-existent Health Record	HealthRecord Controller	HealthRecord.test .js	UT6
R7	Add Doctor	Doctor Controller	Doctor.test.js	UT7
R8	Get All Doctors	Doctor Controller	Doctor.test.js	UT8
R9	Get Doctor by ID	Doctor Controller	Doctor.test.js	UT9
R10	Get Doctor by Invalid ID	Doctor Controller	Doctor.test.js	UT10
R11	Update Doctor by ID	Doctor Controller	Doctor.test.js	UT11
R12	Delete Doctor by ID	Doctor Controller	Doctor.test.js	UT12
R13	Delete Doctor by Invalid ID	Doctor Controller	Doctor.test.js	UT13
R14	Add Appointment	Appointment Controller	Appointment.test. js	UT14
R15	Get All Not-Approved Appointments	Appointment Controller	Appointment.test. js	UT15

R16	Get All Approved Appointments	Appointment Controller	Appointment.test.js	UT16
R17	Update Appointment by ID	Appointment Controller	Appointment.test.js	UT17
R18	Delete Appointment by ID	Appointment Controller	Appointment.test.js	UT18
R19	Add Jailor	Jailor Controller	Jailor.test.js	UT19
R20	Get All jailors	Jailor Controller	Jailor.test.js	UT20
R21	Get Jailor by ID	Jailor Controller	Jailor.test.js	UT21
R22	Update Jailor Information	Jailor Controller	Jailor.test.js	UT22
R23	Delete Jailor	Jailor Controller	Jailor.test.js	UT23
R24	Get Jailor by Invalid ID	Jailor Controller	Jailor.test.js	UT24
R25	Add Jailor without Field	Jailor Controller	Jailor.test.js	UT25
R26	Add Jailor From Frontend	Jailor Controller	Jailor.test.js	UT26
R27	Validate Empty Fields	Jailor Controller	Jailor.test.js	UT27
R28	Add+Get Jailor	Jailor Controller	Jailor.test.js	UT28
R29	Update Jailor Details	Jailor Controller	Jailor.test.js	UT29
R30	Add Jailors Underload	Jailor Controller	Jailor.test.js	UT30
R31	Get All Jailors Speed	Jailor Controller	Jailor.test.js	UT31
R32	User Registration	User Controller	User.test.js	UT32
R33	User Login	User Controller	User.test.js	UT33
R34	User Deletion	User Controller	User.test.js	UT34
R35	Invalid User	User Controller	User.test.js	UT35

R36	Register User through Frontend	User Controller	User.test.js	UT36
R37	Login through UI	User Controller	User.test.js	UT37
R38	Register and Login Flow	User Controller	User.test.js	UT38
R39	Delete and Fetch User	User Controller	User.test.js	UT39
R40	Bulk User Registration	User Controller	User.test.js	UT40
R41	Multiple Concurrent Logins	User Controller	User.test.js	UT41
R42	Fetch All Visit Records	Visit Controller	Visit.test.js	UT42
R43	Get Visit By Valid ID	Visit Controller	Visit.test.js	UT43
R44	Add a new Visit	Visit Controller	Visit.test.js	UT44
R45	Update Existing Visit	Visit Controller	Visit.test.js	UT45
R46	Delete Existing Visit	Visit Controller	Visit.test.js	UT46
R47	Delete Non Existing Visit	Visit Controller	Visit.test.js	UT47
R48	Add Visit Through Form	Visit Controller	Visit.test.js	UT48
R49	Empty Field Validation	Visit Controller	Visit.test.js	UT49
R50	Add and Fetch via API	Visit Controller	Visit.test.js	UT50
R51	Update+Verify Update	Visit Controller	Visit.test.js	UT51
R52	Add Multiple Visits	Visit Controller	Visit.test.js	UT52
R53	GET Performance	Visit Controller	Visit.test.js	UT53

R54	Fetch All Visitors Records	Visitor Controller	Visitor.test.js	UT54
R55	Get Visitor by Valid ID	Visitor Controller	Visitor.test.js	UT55
R56	Add a new Visitor	Visitor Controller	Visitor.test.js	UT56
R57	Update Existing Visitor	Visitor Controller	Visitor.test.js	UT57
R58	Delete Existing Visitor	Visitor Controller	Visitor.test.js	UT58
R59	Delete Non Existing Visitor	Visitor Controller	Visitor.test.js	UT59
R60	Add Visitor through Form	Visitor Controller	Visitor.test.js	UT60
R61	Empty Field Validation	Visitor Controller	Visitor.test.js	UT61
R62	Add and Fetch Via API	Visitor Controller	Visitor.test.js	UT62
R63	Update+Verify Update	Visitor Controller	Visitor.test.js	UT63
R64	Add multiple Visitors	Visitor Controller	Visitor.test.js	UT64
R65	Get Performance	Visitor Controller	Visitor.test.js	UT65
R66	Add Inmate	Inmate Controller	Inmate.test.js	UT66
R67	Get All Inmates	Inmate Controller	Inmate.test.js	UT67
R68	Get Inmate by ID	Inmate Controller	Inmate.test.js	UT68
R69	Update Inmate by ID	Inmate Controller	Inmate.test.js	UT69
R70	Delete Inmate by ID	Inmate Controller	Inmate.test.js	UT70
R71	Delete Non-existing Inmate	Inmate Controller	Inmate.test.js	UT71
R72	Get Inmate with Invalid ID	Inmate Controller	Inmate.test.js	UT72

7.3 Conclusion

The core goals identified at the project's onset have been effectively met by the design and testing of the prison management system. We have verified system reliability and performance via careful unit testing of main components such as health record management, appointment scheduling, doctor management, and inmate management. Every requirement has been followed via design, code, and testing stage, therefore checking that the system conforms around the specified Software Requirements Specification (SRS).

By automatically handling structured data and automating manual activities, the system increases operational efficiency in prison management. The comprehensive testing approach chosen validates the system's readiness for release and guarantees it satisfies real-world constraints and user expectations.

The original project goals are matched to the system features created and checked on the table below.

Objectives vs. Functionalities Table

Objective ID	Objective Description	Functionality Provided
OBJ1	Manage inmate records effectively.	Use the Inmate module to add, update, delete, and recover inmate data.
OBJ2	Keep medical and health records of inmates.	Record and recover health data via the Health Records module.
OBJ3	Empower prison physicians to handle and schedule appointments.	View and add appointments through Appointment module.
OBJ4	Allow addition and management of prison doctors	For doctors within the Doctor module, permit the addition and control of prison physicians CRUD operations.
OBJ5	Provide categorized access to inmate status (current, released)	Retrieve inmate data based on status via specialized APIs. Offer classified access to inmate status (current, released)
OBJ6	Make certain that all operations are checked and verified	Comprehensive test coverage across all major functional units (18 test cases)

7.4 Future Work

The existing system can perform its specified scope and goal quite well, but there are aspects that might be improved or added in future versions:

Additional Features:

- **Biometric Authentication:** Fingerprint or facial recognition as safe log in for utilization of prison staff and recognition of visitors.
- **Real-time Notifications:** SMS or email alert for authorized visitor request, medical visits, or even emergency alerts.
- **Audit Trail Module:** Record all system modifications (add/update/delete activities) for transparency and security audit reasons.

Performance-and-Optimization:

- **Database Optimization:** Include advanced index and cache technology to generate faster response to huge volumes of data.
- **Load Balancing:** Employ load balancing to manage concurrency of multiple user sessions.

Scalability Improvements:

- **Cloud Deployment:** It deploys the system on a cloud infrastructure elastic on demand such as AWS or Azure for real-time access from multiple locations.
- **Multi-Tenancy:** It will allow the system to be accessed by more than one prison within the same region or country from a single location.

User Interface Features:

- **Mobile App Version:** The mobile application second version allows the application to be easily used on the move by staff and prison officers.
- **Multilingual Capability:** Translate options to allow the interface to be utilized by more users.

Advanced Analytics and AI:

- **Health Predictive Monitoring:** Analyze prisoner medical history using AI and predict an ailment in health.
- **Behavioral Analysis:** Include in CCTV or RFID streams to track prisoners' activity and detect anomalous activity.

References

- GeeksforGeeks. "Unified Modeling Language (UML) Class Diagrams." Internet: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/, Mar. 5, 2023 [Accessed: Dec. 7, 2024].
- GeeksforGeeks. "Unified Modeling Language (UML) Sequence Diagrams." Internet: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/, Oct. 10, 2024 [Accessed: Dec. 7, 2024].
- GeeksforGeeks. "State Transition Diagram for an ATM System." Internet: https://www.geeksforgeeks.org/state-transition-diagram-for-an-atm-system, Nov. 11, 2022 [Nov. 20, 2024].
- Visual Paradigm. "Sequence Diagram Tutorial: Complete Guide with Examples." Internet:https://www.visual-paradigm.com/guide/uml-unified-modeling-language/sequence-diagram/, [Accessed: Dec. 7, 2024].
- Lucidchart. "What is UML? A Beginner's Guide to Unified Modeling Language." Internet: https://www.lucidchart.com/pages/what-is-uml, [Accessed: Dec. 7, 2024].
- Dribbble. "UI Style Guide." Internet: https://dribbble.com/tags/ui-style-guide, [Nov. 23, 2024].
- E. Ahishakiye, D. Taremwa, and E. O. Omulo, "A Secure Web Based Records Management System for Prisons: A Case of Kisoro Prison in Uganda," *International Journal of Computer (IJC)*, vol. 24, no. 1, 2017.
- J. Yang, "Big Data Technology and Prison Management Analysis," in *Proc. 2021 International Conference on Big Data Analysis and Computer Science (BDACS 2021)*, 2021. https://doi.org/10.1109/BDACS53596.2021.00016
- M. Bennett. "Benefits of prison management systems." Internet: https://www.m2sys.com/blog/e-governance/benefits-of-prison-management-systems/, May 15, 2024 [Oct. 18, 2024].
- Figma. "Figma: The Collaborative Interface Design Tool." Internet: https://www.figma.com/, [Accessed: Dec. 7, 2024].
- Apache JMeter. "Apache JMeter™." Internet: https://jmeter.apache.org/, [Accessed: Dec. 7, 2024].
- Docker. "What is Docker?" Internet: https://www.docker.com/resources/what-container/, [Accessed: Dec. 7, 2024].
- GitHub. "GitHub: Where the world builds software." Internet: https://github.com/, [Accessed: Dec. 7, 2024].

Appendix A

Use case Description Template

Appendix-A Use Case Description(Fully Dressed Format)

Table A- 1Show the detail use case template and example

Use Case ID:	UC-1
Use Case Name:	Login
Actors:	Administrator, Visitors, Prison Staff, Prisoner
Description:	This use case enables a user to log in to the system.
Trigger:	User attempts to log in.
Preconditions:	The user must be a registered user in the system.
Postconditions:	The user is successfully logged in and enters the system.
Normal Flow:	User opens the login page. Inputs their password and login ID. The system verifies the login credentials and grants the user access.
Alternative Flows:	The system notifies the user of an error and requests that they try again if the entered data is invalid.
Business Rules:	Account locks after three unsuccessful login attempts.
Assumptions:	User knows their login credentials.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-2
Use Case Name:	Register Prisoner
Actors:	Administrator
Description:	This use case is initiated when the administrator registers a new prisoner.
Trigger:	Administrator chooses to register a prisoner.
Preconditions:	Administrator must be logged in. Prisoner information must be available.
Postconditions:	Prisoner is registered with a unique ID and information is saved.

Normal Flow:	Administrator logged into the system. Administrator selects the 'Register Prisoner' option. Administrator enters the prisoner's details (name, DOB, crime details etc.). System checks and stores the details. The administrator assigns a unique ID to prisoner.
Alternative Flows:	If any mandatory information is missing, the system will ask the administrator to provide the relevant information before proceeding.
Business Rules:	Duplicate entries are not allowed.
Assumptions:	Administrator has all necessary prisoner details.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-3
Use Case Name:	View Prisoner Record
Actors:	Administrator
Description:	This use case enables the administrator to view prisoner information.
Trigger:	Administrator selects the option to view prisoner records.
Preconditions:	Administrator must be logged in. Prisoner must be registered.
Postconditions:	Prisoner record is displayed.
Normal Flow:	Administrator logged into the system. Administrator selects the View Prisoner Record . The administrator enters the prisoner's ID, and the system shows the prisoner's details.
Alternative Flows:	If the prisoner ID entered is invalid or not found, the system displays an error message.
Business Rules:	Only authorized administrators can view records.
Assumptions:	Prisoner record exists in the system.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-4
Use Case Name:	Update Prisoner Record
Actors:	Administrator
Description:	This use case enables the administrator to update prisoner information.
Trigger:	Administrator selects the Update Prisoner Record option.
Preconditions:	Administrator must be logged in and the prisoner must be registered.
Postconditions:	Prisoner record is updated successfully.
Normal Flow:	Administrator logged into the system. Administrator selects the Update Prisoner Record. The administrator enters the prisoner's ID, and the system shows the prisoner's details. The admin Officer makes the required changes. The system updates the record
Alternative Flows:	If the prisoner ID entered is invalid or not found, the system displays an error message.
Business Rules:	Only administrators with edit privileges can perform updates.
Assumptions:	All changes are logged by the system.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-5
Use Case Name:	Remove Prisoner Record
Actors:	Administrator
Description:	This use case allows the administrator to remove a prisoner's record.
Trigger:	Administrator selects the Remove Prisoner Record option.
Preconditions:	Administrator must be logged in and prisoner must be registered.
Postconditions:	Prisoner record is deleted from the system.

Normal Flow:	Administrator logged into the system.The. Administrator selects the Remove Prisoner Record . The administrator enters the prisoner's ID, and the system shows the prisoner's details. The system removes the record.
Alternative Flows:	If the prisoner ID entered is invalid or not found, the system displays an error message.
Business Rules:	Record deletion is irreversible.
Assumptions:	Administrator has delete permissions.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-6
Use Case Name:	Allocate Cells
Actors:	Administrator
Description:	Allows administrator to assign a prisoner to a cell.
Trigger:	Administrator selects Allocate Cells option.
Preconditions:	Prisoner must be registered and cells must be available.
Postconditions:	Prisoner is assigned to a cell and system updates the record.
Normal Flow:	Administrator logged into the system. The administrator enters the prisoner's ID, and the system retrieves available cells. The administrator selects a cell, and the system assigns the prisoner to it.
Alternative Flows:	If no available cells are found, the system notifies the administrator and suggests alternative actions like transferring the prisoner
Business Rules:	Only vacant cells can be assigned.
Assumptions:	System maintains up-to-date cell availability.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-7
Use Case Name:	Schedule Duties
Actors:	Administrator
Description:	Administrator assigns work shifts to staff members.
Trigger:	Administrator selects Schedule Duties option.
Preconditions:	Staff members must be registered.
Postconditions:	Shift schedule is saved in the system.
Normal Flow:	Administrator logged into the system. The administrator enters the staff member's ID, selects a shift, and assigns it.The system saves the shift schedule.
Alternative Flows:	If the staff ID is invalid or not found, display an error message.
Business Rules:	Shifts cannot overlap.
Assumptions:	All staff details are current.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-8
Use Case Name:	View Complaints
Actors:	Administrator
Description:	Enables administrator to view prisoner complaints.
Trigger:	Administrator selects View Complaints option.
Preconditions:	Administrator must be logged in.
Postconditions:	Complaints are reviewed and logged.
Normal Flow:	Administrator logged into the system. The system retrieves the list of complaints, and the administrator can view each one individually.
Alternative Flows:	If no complaints are found, the system informs the administrator that there are no pending complaints to review.
Business Rules:	Only unreviewed complaints are shown.
Assumptions:	Complaints are submitted through the system.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-9
Use Case Name:	Manage Visit Requests
Actors:	Administrator
Description:	Enables administrator to manage visit requests.
Trigger:	Administrator selects Manage Visit Requests option.
Preconditions:	User must be registered.
Postconditions:	Visit requests are processed.
Normal Flow:	Administrator logged into the system. Administrator is presented with the option ‘manage visit requests’. Upon clicking this option, the administrator can further choose between two options "Manage Visitor Visit Request" and "Manage Prisoner Visit Request".
Alternative Flows:	If visit requests do not show any further option, the system displays an error message.
Business Rules:	Visit limits are enforced.
Assumptions:	Requests are submitted in system.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-10
Use Case Name:	Manage Prisoner Visit Request
Actors:	Administrator
Description:	Allows administrator to process prisoner visit requests.
Trigger:	Administrator selects Prisoner Visit Request option.
Preconditions:	User must be registered.
Postconditions:	Request is approved or denied.
Normal Flow:	Administrator logged into the system. The administrator selects “Manage Prisoners Visit Request”. System display’s all pending requests. After choosing a specific request, the administrator reviews its details and decides to approve or deny it.

Alternative Flows:	If no visit requests are found, the system informs the administrator that there are no pending visit requests to review.
Business Rules:	Only one visitor per visit rule enforced.
Assumptions:	Requests are visible after officer permission.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-11
Use Case Name:	Manage Visitor Visit Requests
Actors:	Administrator
Description:	Allows administrator to process visitor visit requests.
Trigger:	Administrator selects Visitor Visit Request option.
Preconditions:	User must be registered.
Postconditions:	Request is approved or denied.
Normal Flow:	Administrator logged into the system. The administrator selects “Manage Visitors Visit Request”. System display’s all pending requests. After choosing a specific request, the administrator reviews its details and decides to approve or deny it.
Alternative Flows:	If no visit requests are found, the system informs the administrator that there are no pending visit requests to review.
Business Rules:	Visit time restrictions enforced.
Assumptions:	Visitors are pre-verified.

Table A- 1Show the detail use case template and example

Use Case ID:	UC-12
Use Case Name:	Manage Medical Requests
Actors:	Administrator
Description:	Administrator manages medical requests approved by staff.
Trigger:	Administrator selects Manage Medical Requests option.
Preconditions:	User must be registered.
Postconditions:	Medical request is scheduled.
Normal Flow:	Administrator logged into the system. The administrator selects “Manage Medical Request”. System display’s all requests permitted by staff officer After choosing a specific request, the administrator reviews its details and schedules it.
Alternative Flows:	If the administrator finds an invalid or incomplete request permitted by a staff officer, an error message is displayed by the system.
Business Rules:	Only medically approved requests are processed.
Assumptions:	Medical data is available in the system.

Table A- 1Show the detail use case template and example

Use Case ID	UC-13
Use Case Name	View Duties Schedule
Actors	Prison staff
Description	Staff view their scheduled work shifts.
Trigger	Prison staff logs into the system and selects the "Duties Schedule" option.
Preconditions	Prison staff must be logged into the system, and the administrator must have already scheduled the shifts.
Postconditions	Prison staff will be able to view their duties and schedules.
Normal Flow	The prison officer logs into the system and selects the “Duties Schedule” option. The system displays the duties and shifts for the week. The prison officer views the schedule.

Alternative Flows	If no duties are scheduled, the system informs the prison staff. If the system cannot retrieve the schedule, an error message is displayed.
Business Rules	Only authenticated prison staff can access the duties schedule. Duties must be scheduled in advance by an administrator for them to be available.
Assumptions	The system is operational, network connectivity is available, and the staff has valid login credentials.

Table A- 1Show the detail use case template and example

Use Case ID	UC14
Use Case Name	Assign Work
Actors	Prison staff
Description	Staff assign work tasks to prisoners.
Trigger	Prison staff logs into the system and selects the “Assign Work Assignment” option to assign work to prisoners.
Preconditions	Prison staff has logged into the system. Work assignments are available.
Postconditions	The work assignment is successfully assigned to the prisoner, and the system updates the prisoner’s record with the new work assignment.
Normal Flow	Prison staff logs into the system. Prison staff selects the “Assign Work Assignment” option. The system displays a list of prisoners and available work assignments. Prison staff selects a prisoner and assigns the work assignment. The system updates the prisoner’s record.
Alternative Flows	If no work assignments are available, the system informs the Prison staff.
Business Rules	Only prison staff with the required permissions can assign work tasks to prisoners. The system ensures that only valid work assignments can be selected for a prisoner.
Assumptions	The system assumes that the work assignments have been predefined by an administrator and are available when needed by prison staff.

Table A- 1Show the detail use case template and example

Use Case ID	UC15
Use Case Name	Permit Medical Requests
Actors	Prison staff
Description	Staff review and approve or deny prisoner medical checkup requests.
Trigger	Prison staff logs into the system and selects the “Medical requests” option to review and approve or deny prisoner medical checkup requests.
Preconditions	Prison staff has logged into the system. Medical requests are waiting for approval.
Postconditions	The medical request is either approved or denied, and the system updates the request status accordingly.
Normal Flow	Prison staff logs into the system. Prison staff selects the “Medical requests” option. The system displays a list of pending medical requests. Prison staff reviews the details of each medical request and approves or denies them. The system updates the request status.
Alternative Flows	If a medical request is denied, the system notifies the prisoner.
Business Rules	Only prison staff with appropriate permissions can approve or deny medical requests. The system ensures that only valid medical requests are processed.
Assumptions	The system assumes that all medical requests are entered correctly by prisoners and are awaiting review and approval by the prison staff.

Table A- 1Show the detail use case template and example

Use Case ID	UC16
Use Case Name	Request Visitor
Actors	Prisoner
Description	Allows the prisoner to request a visitor.

Trigger	The prisoner logs into the system and selects the “Request Visitor” option.
Preconditions	The prisoner has logged into the system and has a list of approved visitors.
Postconditions	The request for a visitor is submitted and is pending approval.
Normal Flow	The prisoner logs into the system. The prisoner selects the “Request Visitor” option. The system displays a list of approved visitors. The prisoner selects a visitor and submits the request. The system confirms the submission.
Alternative Flows	If the request is not submitted, the prisoner can try again later.
Business Rules	The prisoner must have a valid list of approved visitors in order to submit a request. The system ensures that only approved visitors are available for selection.
Assumptions	It is assumed that the prisoner is authorized to request a visitor and has already been registered in the system.

Table A- 1Show the detail use case template and example

Use Case ID	UC17
Use Case Name	Request Work Assignment
Actors	Prisoner
Description	The prisoner requests a work assignment.
Trigger	The prisoner logs into the system and selects the “Request Work Assignment” option.
Preconditions	The prisoner has logged into the system, and work assignments are available.
Postconditions	The prisoner’s request is submitted for review by prison staff.
Normal Flow	The prisoner logs into the system. The prisoner selects the “Request Work Assignment” option. The system displays available work assignments. The prisoner selects a work assignment. The prisoner submits the request. The system confirms the submission and sends it to the staff for review.

Alternative Flows	If no work assignments are available, the system informs the prisoner.
Business Rules	The prisoner must be registered in the system to request work assignments. Only available assignments will be displayed to the prisoner. The request will be sent for review by the staff.
Assumptions	The prisoner is authorized to request work assignments and has access to the list of available tasks.

Table A- 1Show the detail use case template and example

Use Case ID	UC18
Use Case Name	Request Medical Checkup
Actors	Prisoner
Description	The prisoner requests a medical checkup or health services.
Trigger	The prisoner logs into the system and selects the “Request Medical Checkup” option.
Preconditions	The prisoner has logged into the system.
Postconditions	The prisoner’s request is submitted for review by prison staff.
Normal Flow	The prisoner logs into the system. The prisoner selects the “Request Medical Checkup” option. The prisoner provides details of their medical issue. The prisoner submits the medical request.
Alternative Flows	If some data is missing, the system gives an error message to the prisoner to complete the report before submission.
Business Rules	Only valid and complete medical requests will be accepted. The prisoner’s medical issue must be accurately described for the request to be processed.
Assumptions	The prisoner has a legitimate medical need and is authorized to request medical checkups through the system.

Table A- 1Show the detail use case template and example

Use Case ID	UC19
Use Case Name	File Complaints
Actors	Prisoner
Description	The prisoner can submit a complaint about their issues.
Trigger	The prisoner logs into the system and selects the “File Complaint” option.
Preconditions	The prisoner has logged into the system.
Postconditions	The complaint is submitted to the system for review by the administrator.
Normal Flow	The prisoner logs into the system. The prisoner selects the “File Complaint” option. The system displays the complaint form. The prisoner writes about their complaint. The prisoner submits the complaint.
Alternative Flows	If the complaint is not submitted, the prisoner can try again later.
Business Rules	Complaints must be submitted with valid content. Incomplete complaints may not be processed.
Assumptions	The prisoner has an issue that justifies filing a complaint.

Table A- 1Show the detail use case template and example

Use Case ID	UC20
Use Case Name	Request Visit
Actors	Visitor
Description	This use case allows the visitor to submit a request for visiting a prisoner.
Trigger	The visitor logs into the system and selects the “Request Visit” option.
Preconditions	The visitor must be registered in the system, and the prisoner they wish to visit must also be registered.
Postconditions	The request for a visit is submitted and is pending approval.

Normal Flow	The visitor logs into the system. The visitor selects the “Request Visit” option. The visitor submits the request. The system confirms the submission.
Alternative Flows	If the request is not submitted, the visitor can try again later.
Business Rules	Only registered visitors and prisoners can make a request. A visitor’s request must be approved by the appropriate authorities before a visit is allowed.
Assumptions	The visitor and the prisoner are both correctly registered in the system.

Table A- 1Show the detail use case template and example

Use Case ID	UC21
Use Case Name	View Request Status
Actors	Visitor
Description	This use case allows a visitor to check the status of their previously submitted visit request.
Trigger	The visitor logs into the system and selects the “View Request Status” option.
Preconditions	The visitor must be registered in the system, and there must be a submitted visit request associated with their account.
Postconditions	The visitor successfully views the status of their visit request.
Normal Flow	The visitor logs into the system. The visitor selects the “View Request Status” option. The system shows whether the request is accepted, denied, or pending.
Alternative Flows	If there are no requests pending, the system shows a message indicating no pending requests.
Business Rules	The visitor can only view requests that have been submitted under their account. The status will be visible only if it has been updated by the authorities.
Assumptions	The visitor has previously submitted a request and the request status has been updated.

Appendix-B

Coding Standards

Appendix-B General Coding Standards & Guidelines

1. Follow a consistent variable naming convention throughout the code. E.g.
    - Snakecase (words are delimited by "_" like: variable_one)
    - Pascalcase (words are delimited by capital letters like: VariableOne)
    - Camelcase (words are delimited by capital letters except the initial word like: variableOne)
    - Hungarian Notation (describes the variable type or purpose at the start of the
    - variable name like: arrDistributeGroup)
2. Use naming that visually describes scope like privateField, Const etc
3. Use read only/immutable when a field's value should not be changed after initialization
4. Use only get, for properties that should not be updated from outside
5. Name functions according to their functionalities.
6. Insert appropriate comments to make the code understandable to any reader. Additionally follow a consistent style to do so. E.g.

```
/* the below function will be used for the addition of two variables*/
int Add(){
//logic of the function
}
```

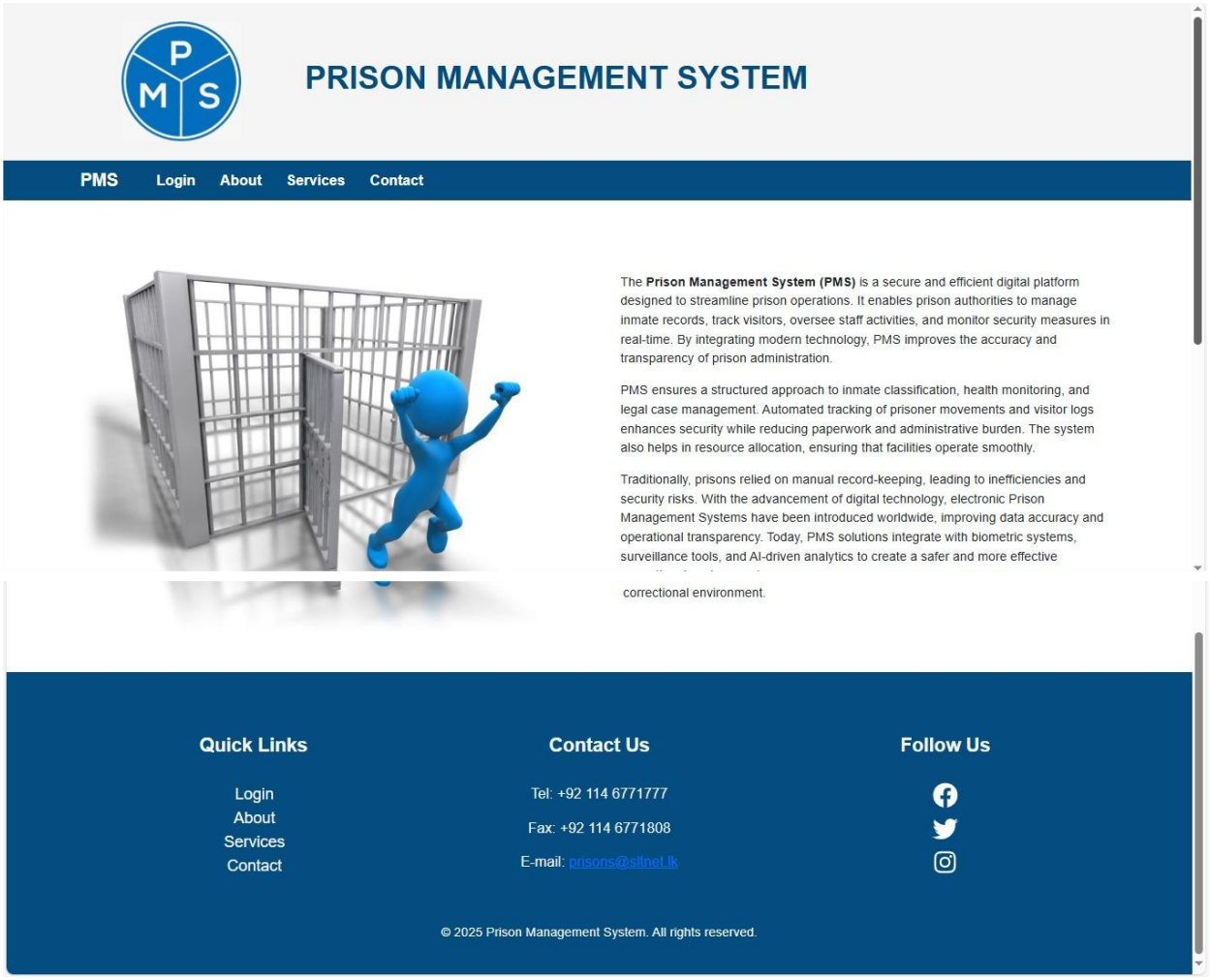
Avoid commenting on obvious things

7. Make use of indentation for indicating the start and end of the control structures along with a clear specification of where the code is between them.
8. Follow consistent naming convention for files and folders.
9. Follow proper structure for classes
10. Group code entities logically into projects/packages/modules/folders
    a. Separate logical layers of application into different modules/services/utilities etc.
    b. User separate files for each class, struct, interface, enum etc. Name of the file and the enclosing entity must be same. E.g., class Employee in Employee.cs/Employee.java
11. Define and use everything within the minimum scope possible
12. Use proper access modifier for all code entities if required
13. Code entities should have maximum cohesion and least coupling possible.
14. Follow DRY law.
    a. Do not repeat code.
    b. A piece of knowledge should exist only in one place within the codebase/application
    c. Reuse code as much as possible
    d. Always write short methods
    e. Single method should not have too many logic, long conditional flow or too many parameters
15. Strictly follow Single Responsibility Principle (SRP) when writing methods, classes, modules, projects, packages, or any other code entities.
16. Write classes and other code entities that are easy to extend without modification.
17. Handle exceptions
18. Log exception and other significant event details
19. Follow a consistent convention for logging all over the application.

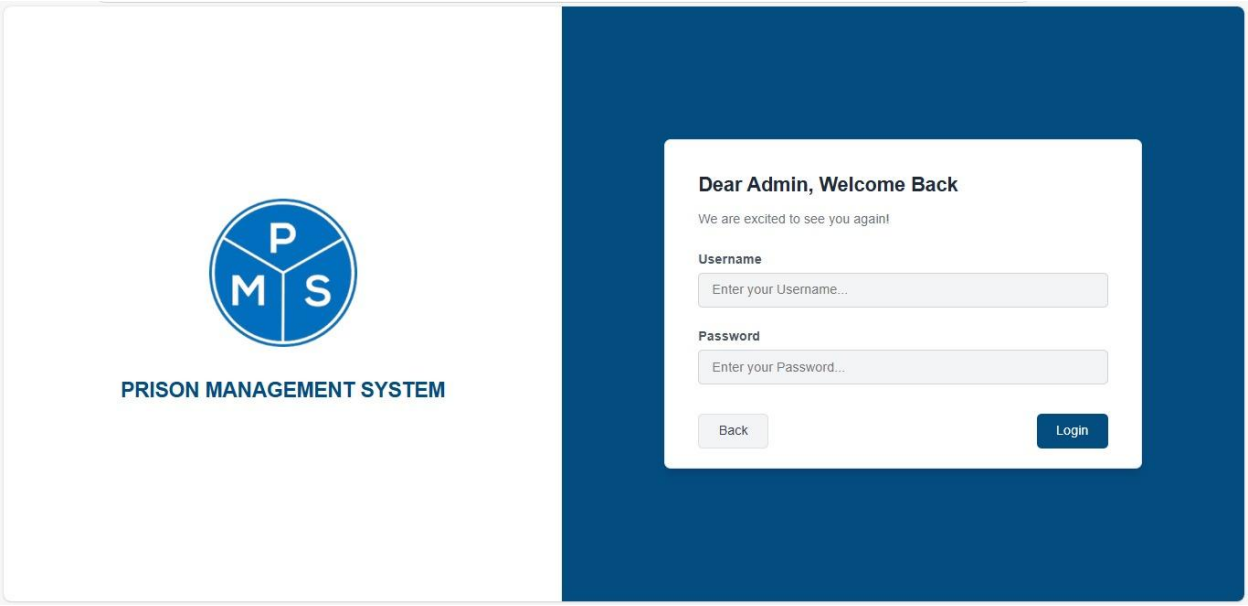
Appendix C

Prototype

Home Page

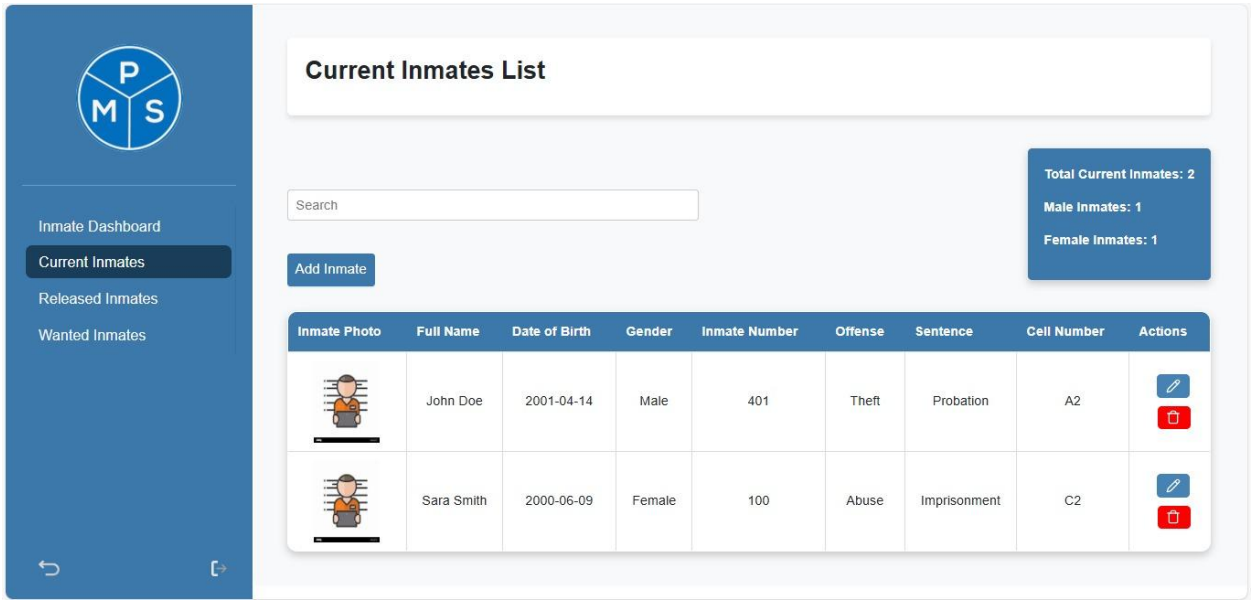

PRISON MANAGEMENT SYSTEM
PMS
Login
About
Services
Contact
The Prison Management System (PMS) is a secure and efficient digital platform designed to streamline prison operations. It enables prison authorities to manage inmate records, track visitors, oversee staff activities, and monitor security measures in real-time. By integrating modern technology, PMS improves the accuracy and transparency of prison administration.
PMS ensures a structured approach to inmate classification, health monitoring, and legal case management. Automated tracking of prisoner movements and visitor logs enhances security while reducing paperwork and administrative burden. The system also helps in resource allocation, ensuring that facilities operate smoothly.
Traditionally, prisons relied on manual record-keeping, leading to inefficiencies and security risks. With the advancement of digital technology, electronic Prison Management Systems have been introduced worldwide, improving data accuracy and operational transparency. Today, PMS solutions integrate with biometric systems, surveillance tools, and AI-driven analytics to create a safer and more effective
correctional environment.
Quick Links
Login
About
Services
Contact
Contact Us
Tel: +92 114 6771777
Fax: +92 114 6771808
E-mail: prisons@sltnet.lk
Follow Us
© 2025 Prison Management System. All rights reserved.

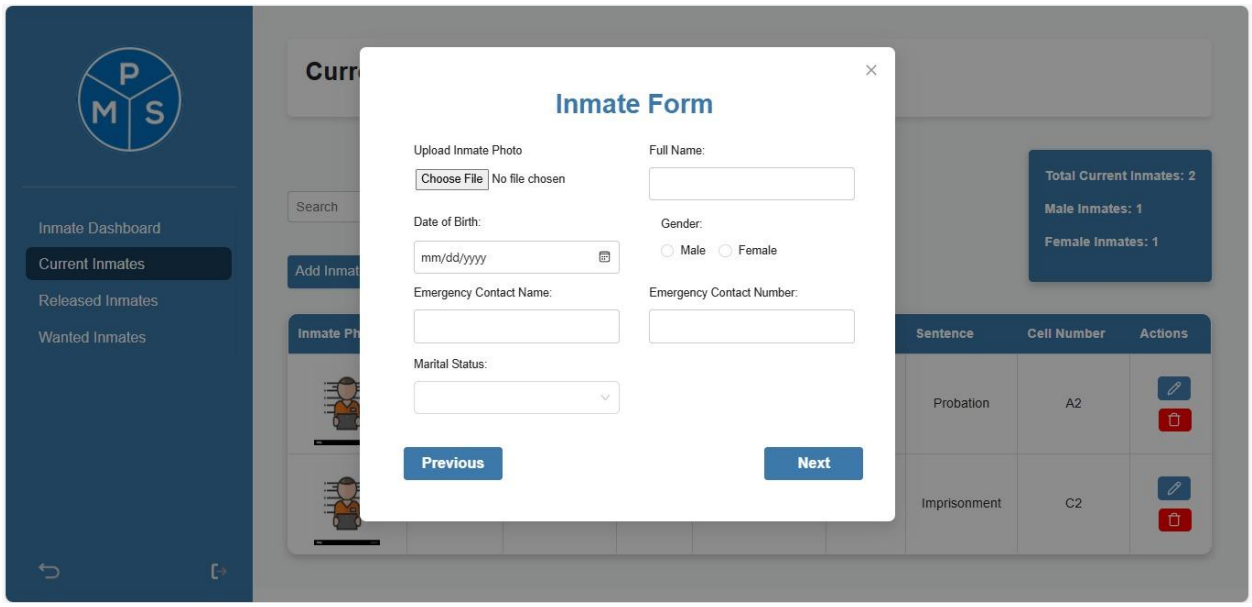
Login Page

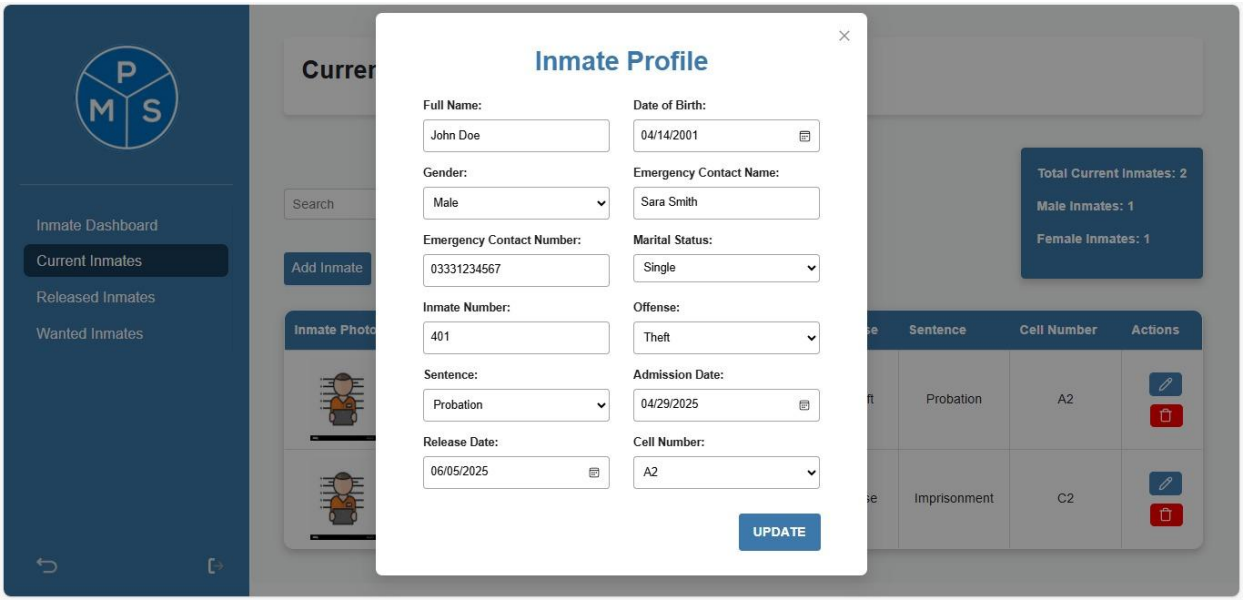


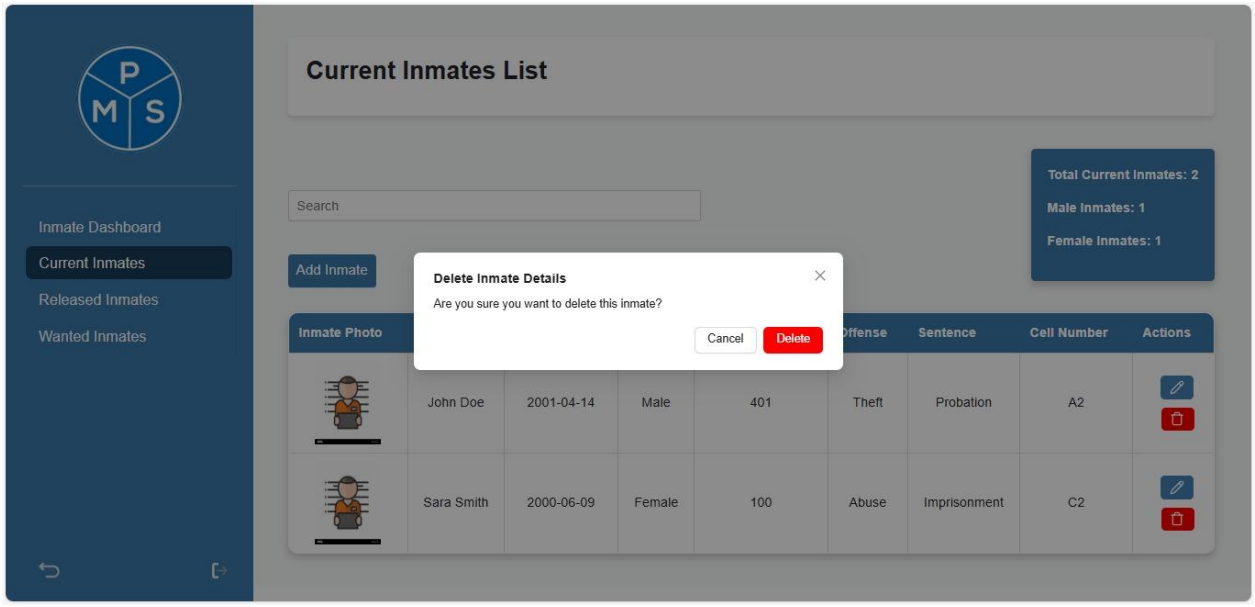
Admin Dashboard

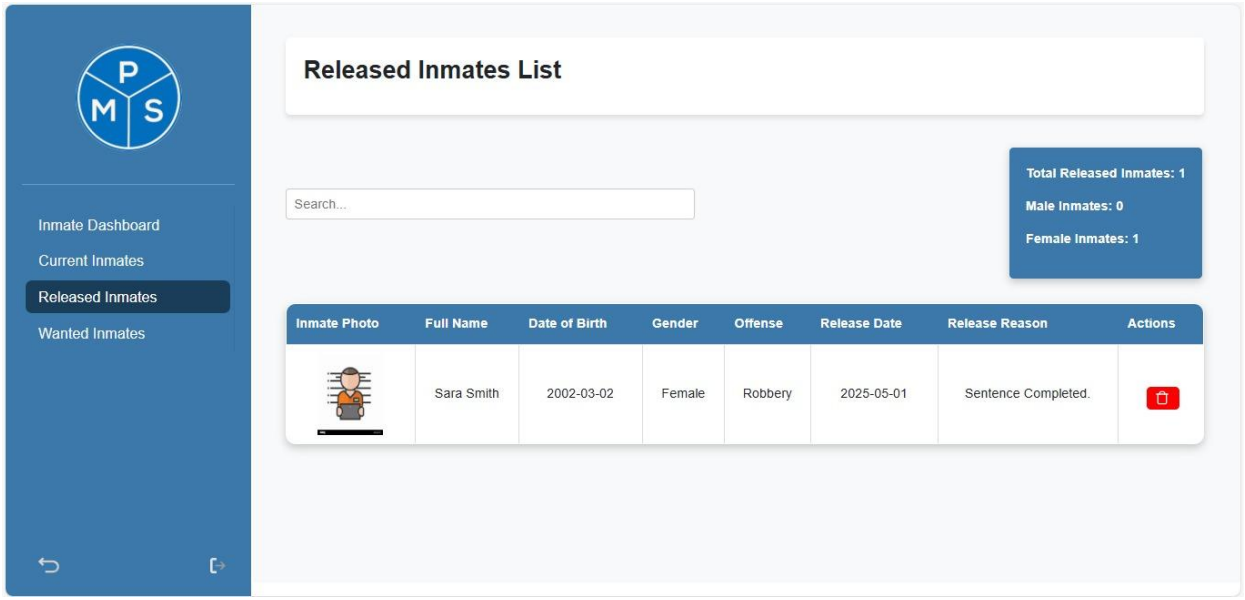
Inmate management

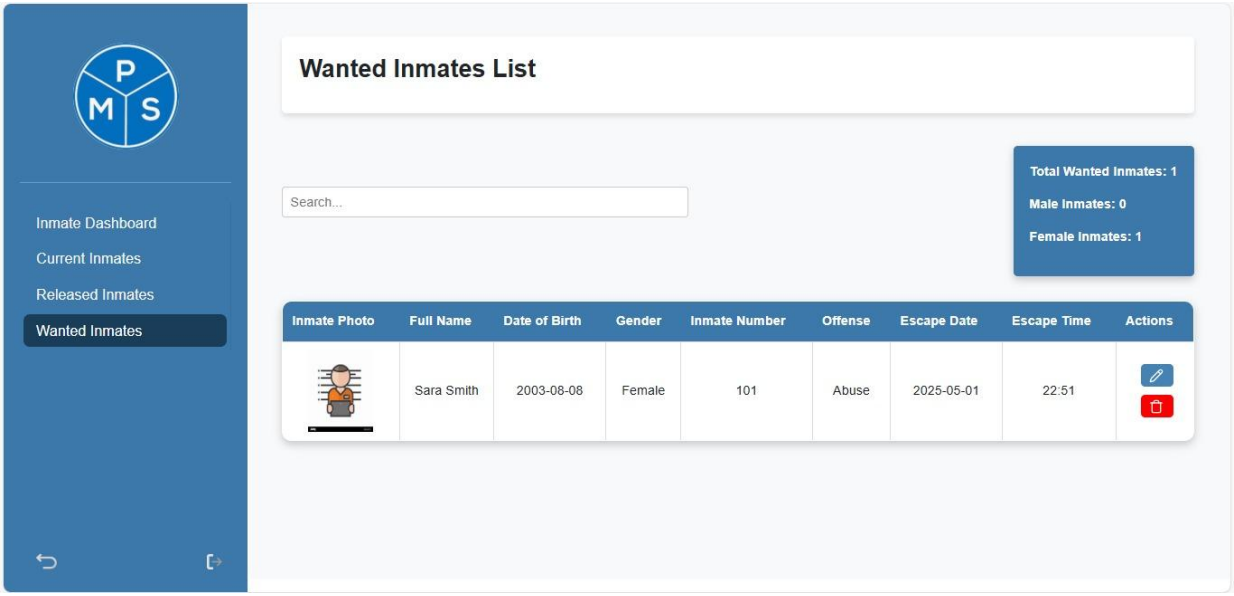



Inmate Dashboard
Current Inmates
Released Inmates
Wanted Inmates
Search
Add Inmat
Total Current Inmates: 2
Male Inmates: 1
Female Inmates: 1
Inmate Form
Upload Inmate Photo
Choose File
No file chosen
Full Name:
Date of Birth:
mm/dd/yyyy
Gender:
Male
Female
Emergency Contact Name:
Emergency Contact Number:
Marital Status:
Previous
Next
Sentence
Cell Number
Actions
Probation
A2
Imprisonment
C2

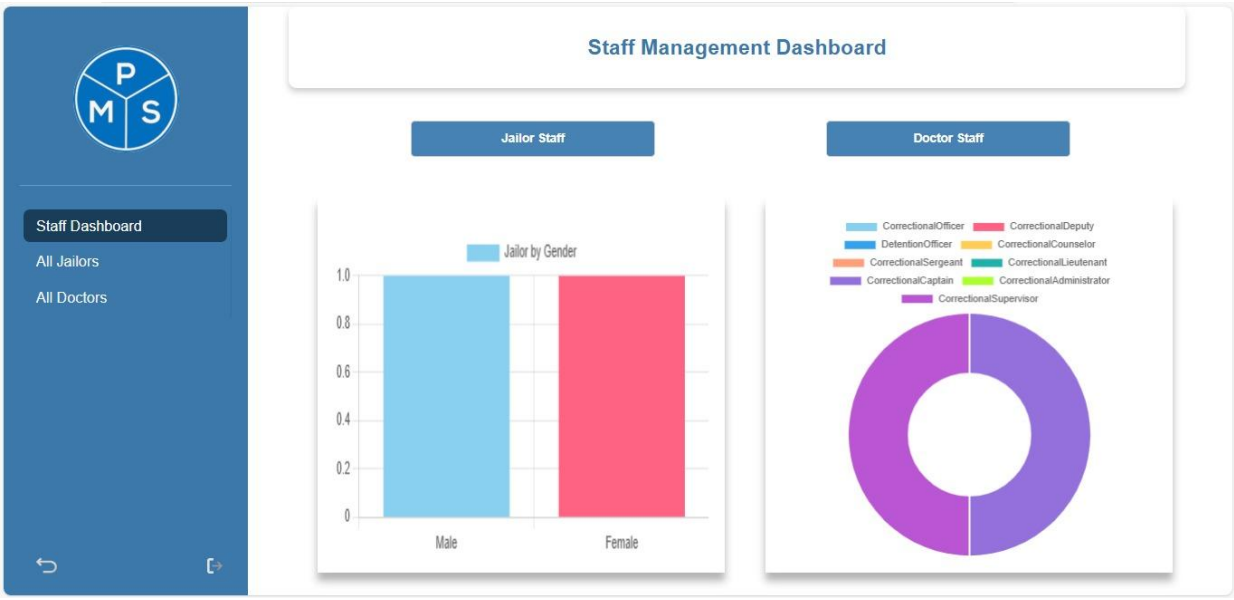

Inmate Dashboard
Current Inmates
Released Inmates
Wanted Inmates
Search
Add Inmate
Total Current Inmates: 2
Male Inmates: 1
Female Inmates: 1
Inmate Profile
Full Name:
John Doe
Date of Birth:
04/14/2001
Gender:
Male
Emergency Contact Name:
Sara Smith
Emergency Contact Number:
03331234567
Marital Status:
Single
Inmate Number:
401
Offense:
Theft
Sentence:
Probation
Admission Date:
04/29/2025
Release Date:
06/05/2025
Cell Number:
A2
UPDATE
Sentence
Cell Number
Actions
Probation
A2
Imprisonment
C2

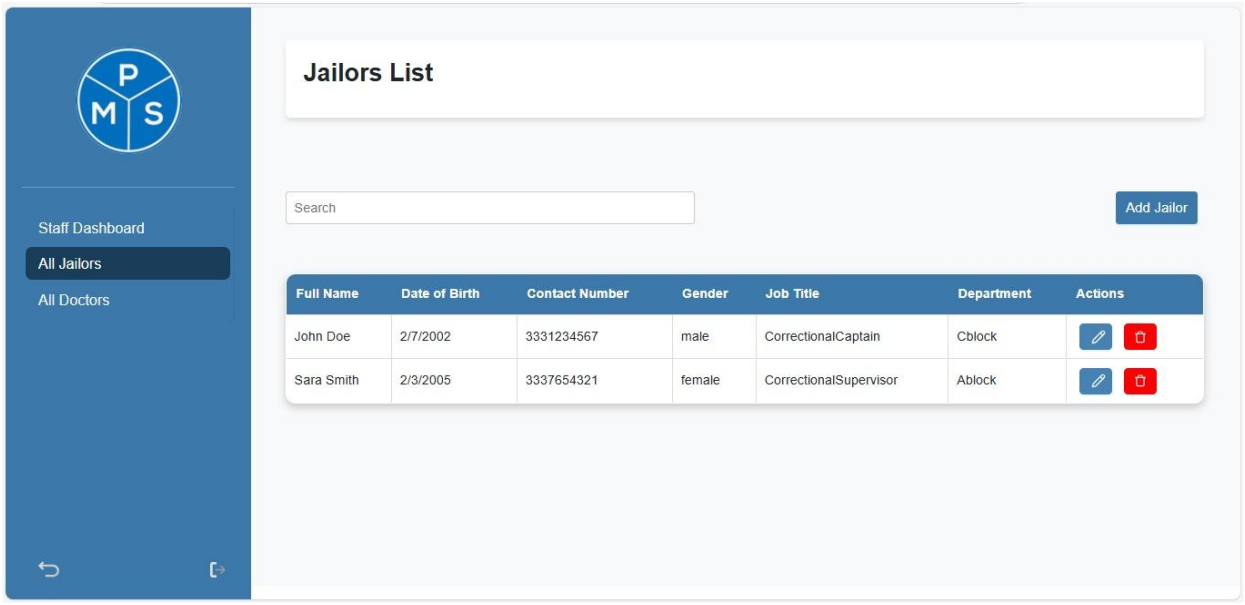

Current Inmates List
Inmate Dashboard
Current Inmates
Released Inmates
Wanted Inmates
Search
Add Inmate
Total Current Inmates: 2
Male Inmates: 1
Female Inmates: 1
Delete Inmate Details
Are you sure you want to delete this inmate?
Cancel
Delete
Inmate Photo
Sentence
Cell Number
Actions
John Doe 2001-04-14 Male 401 Theft Probation A2
Sara Smith 2000-06-09 Female 100 Abuse Imprisonment C2

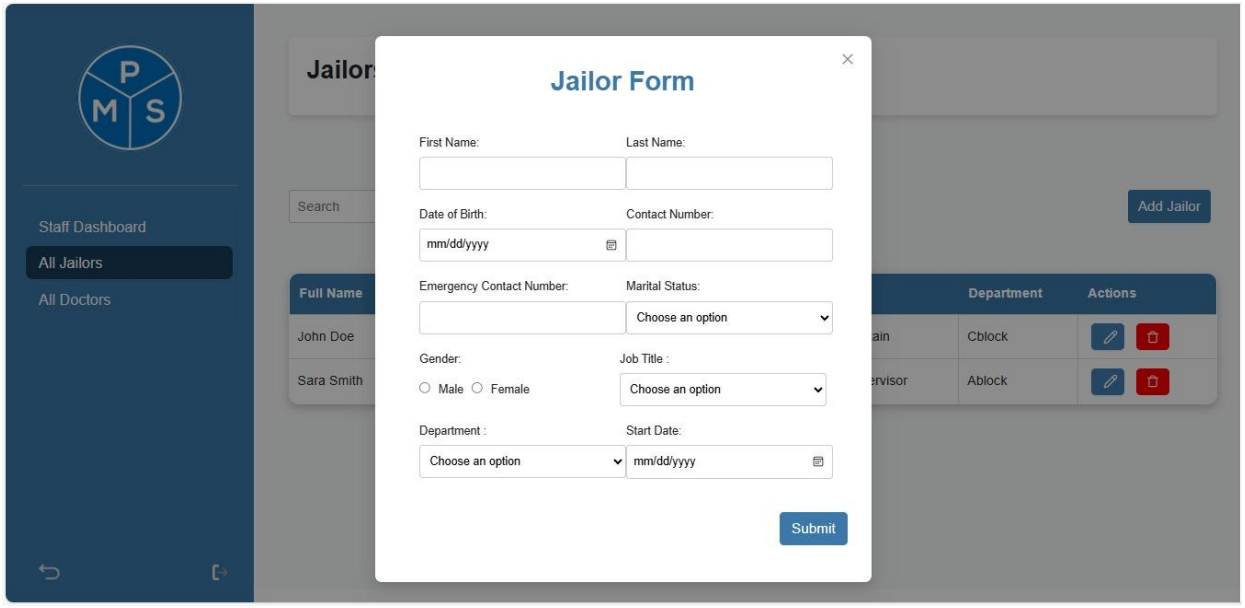

Released Inmates List
Inmate Dashboard
Current Inmates
Released Inmates
Wanted Inmates
Search...
Total Released Inmates: 1
Male Inmates: 0
Female Inmates: 1
Inmate Photo
Full Name
Date of Birth
Gender
Offense
Release Date
Release Reason
Actions
Sara Smith 2002-03-02 Female Robbery 2025-05-01 Sentence Completed.

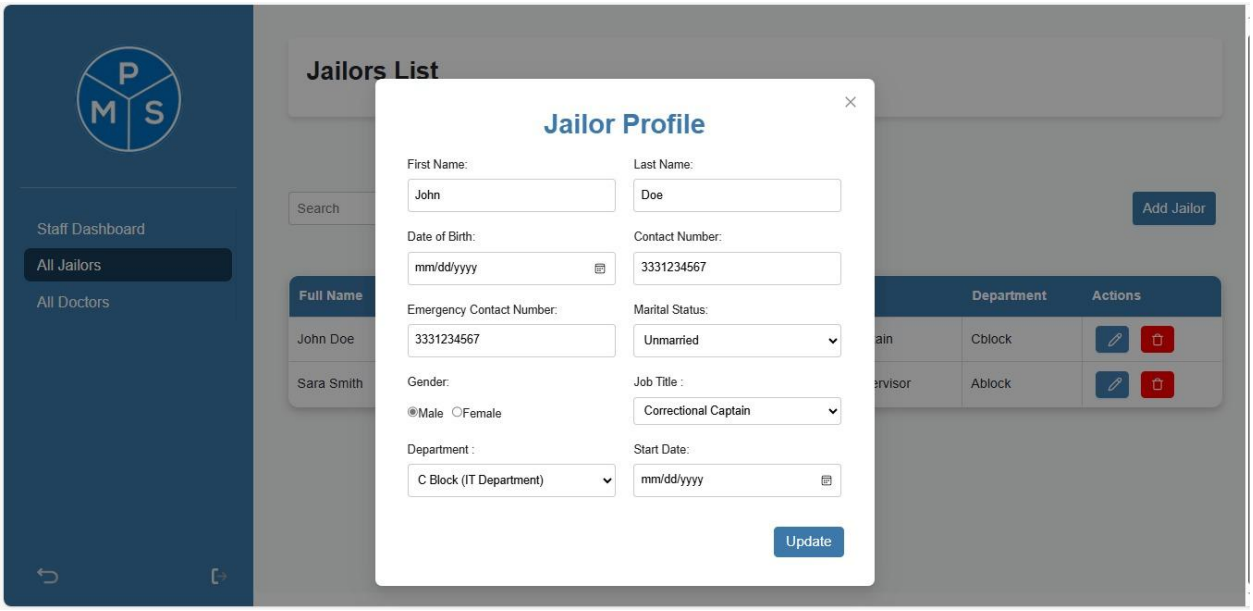


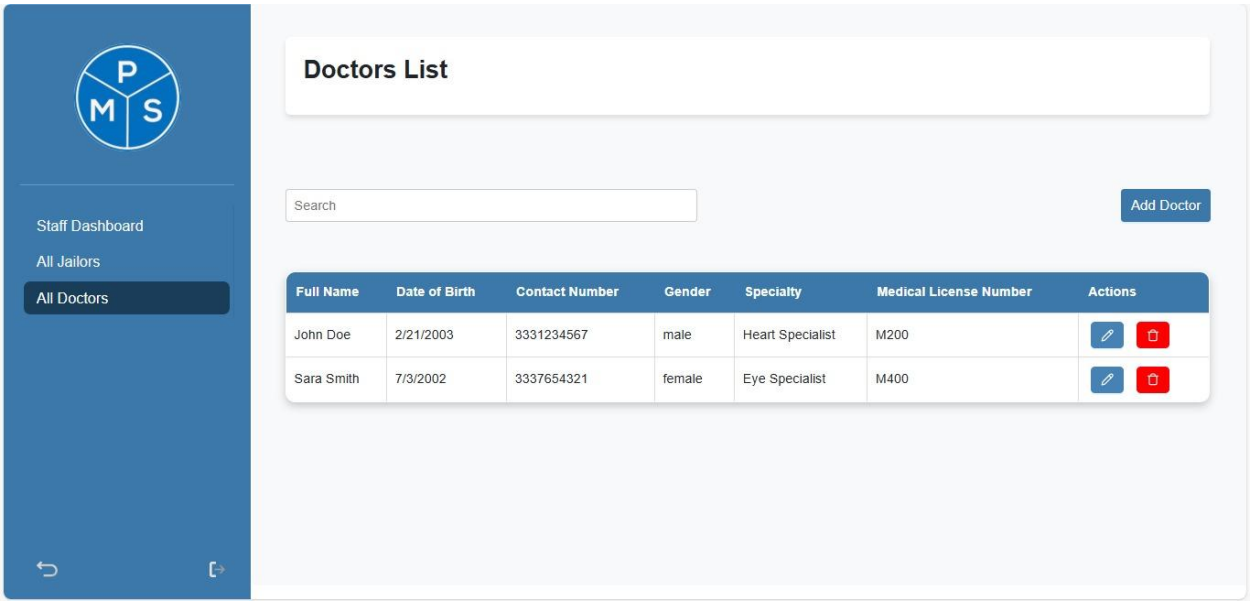
Staff management

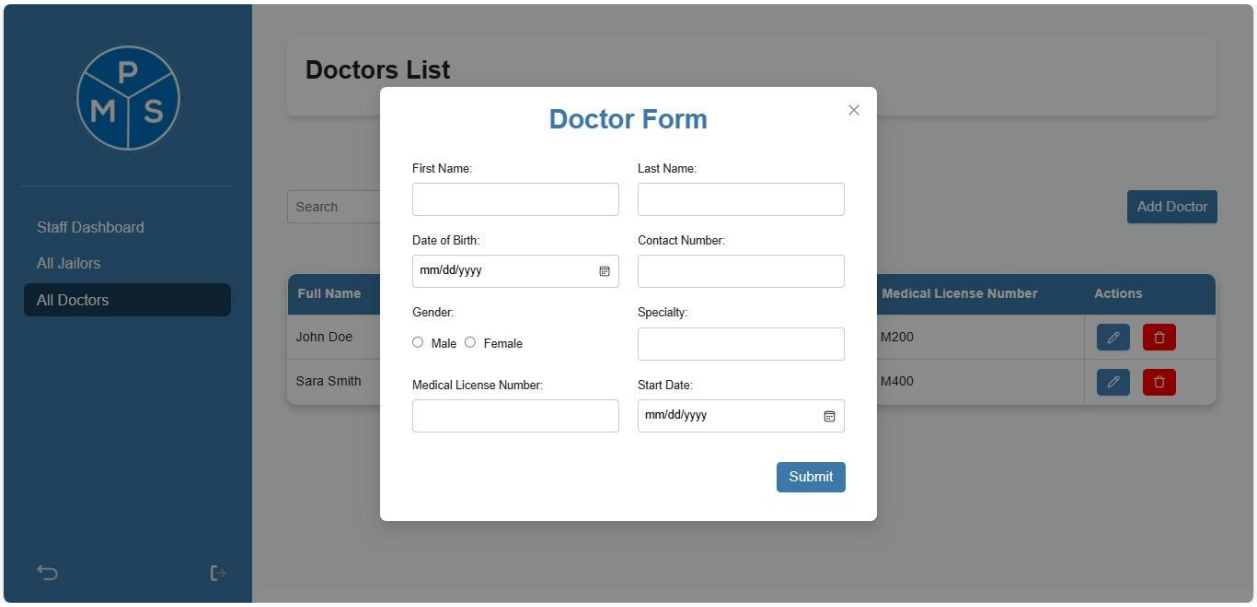


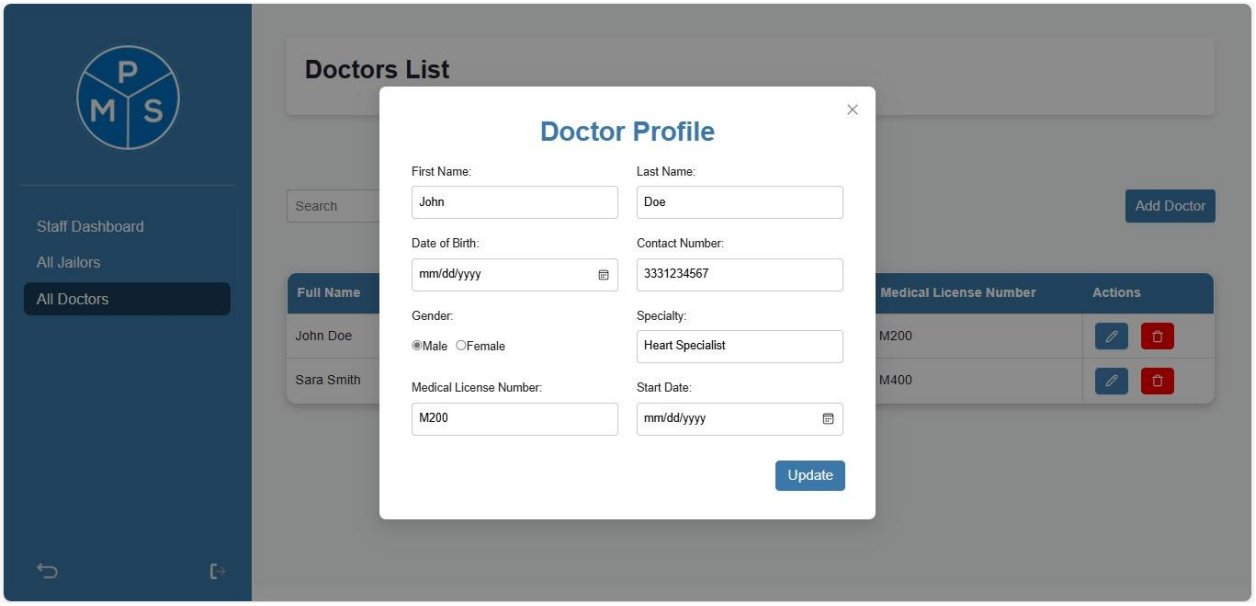

Jailors List
Search
Add Jailor
Staff Dashboard
All Jailors
All Doctors
Full Name
Date of Birth
Contact Number
Gender
Job Title
Department
Actions
John Doe
2/7/2002
3331234567
male
CorrectionalCaptain
Cblock
Sara Smith
2/3/2005
3337654321
female
CorrectionalSupervisor
Ablock


Jailor Form
First Name:
Last Name:
Date of Birth:
mm/dd/yyyy
Contact Number:
Emergency Contact Number:
Marital Status:
Choose an option
Gender:
Male
Female
Job Title :
Choose an option
Department :
Choose an option
Start Date:
mm/dd/yyyy
Submit

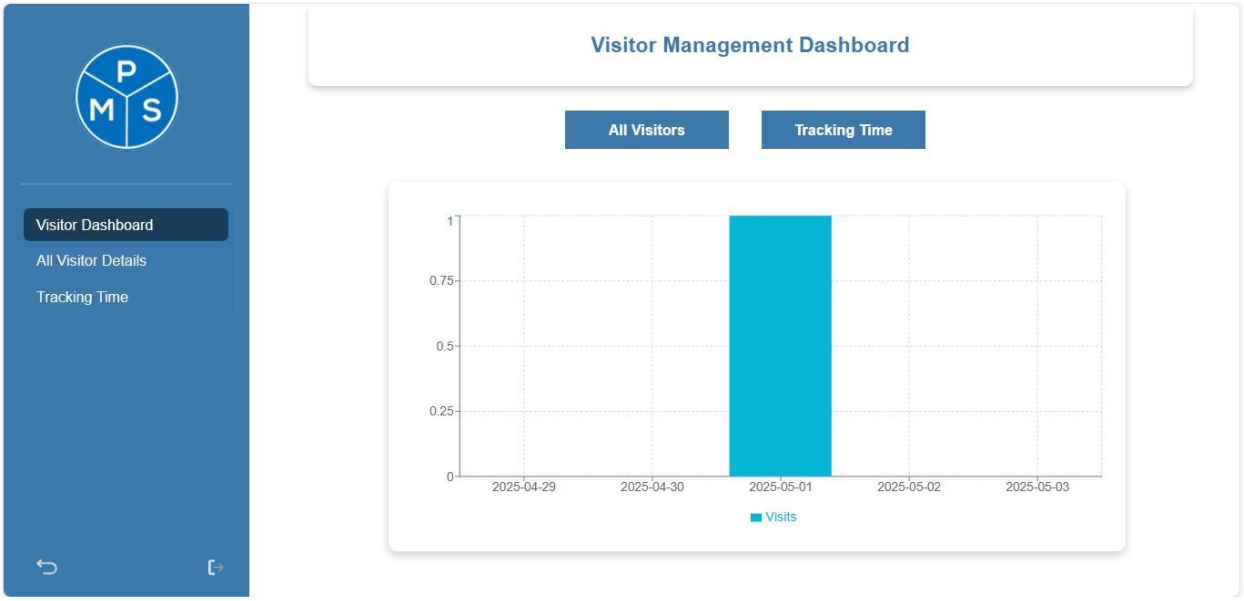

Jailors List
Staff Dashboard
All Jailors
All Doctors
Search
Add Jailor
Full Name
Department
Actions
John Doe
Cblock
Sara Smith
Ablock
Jailor Profile
First Name:
John
Last Name:
Doe
Date of Birth:
mm/dd/yyyy
Contact Number:
3331234567
Emergency Contact Number:
3331234567
Marital Status:
Unmarried
Gender:
Male
Female
Job Title :
Correctional Captain
Department :
C Block (IT Department)
Start Date:
mm/dd/yyyy
Update

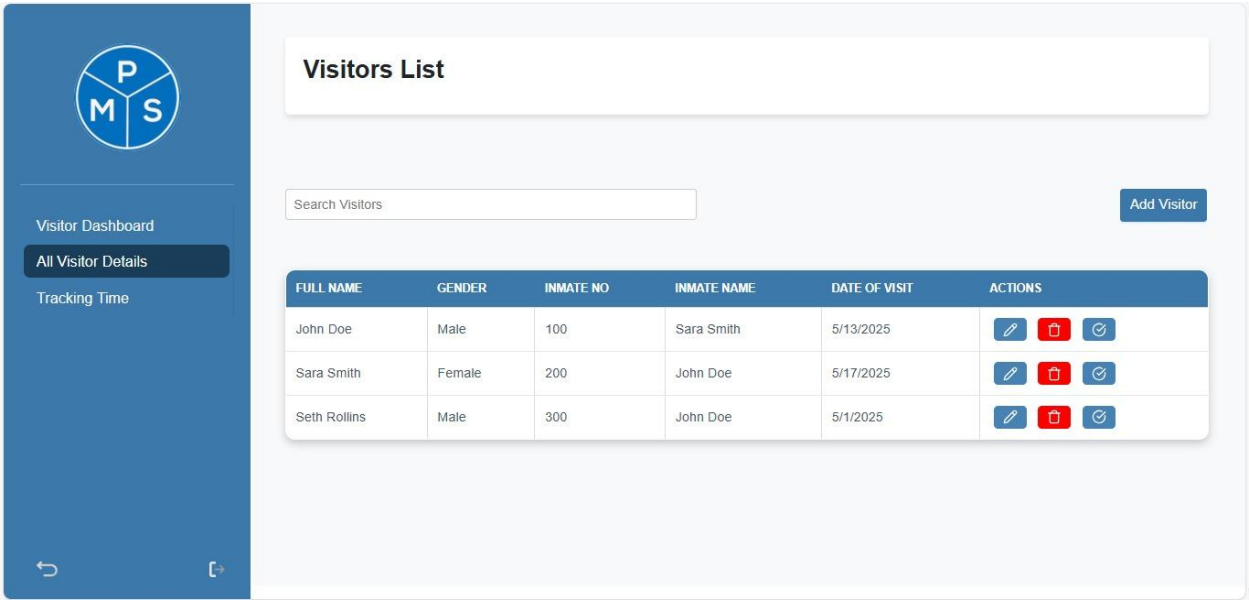

Doctors List
Staff Dashboard
All Jailors
All Doctors
Search
Add Doctor
Full Name
Date of Birth
Contact Number
Gender
Specialty
Medical License Number
Actions
John Doe
2/21/2003
3331234567
male
Heart Specialist
M200
Sara Smith
7/3/2002
3337654321
female
Eye Specialist
M400

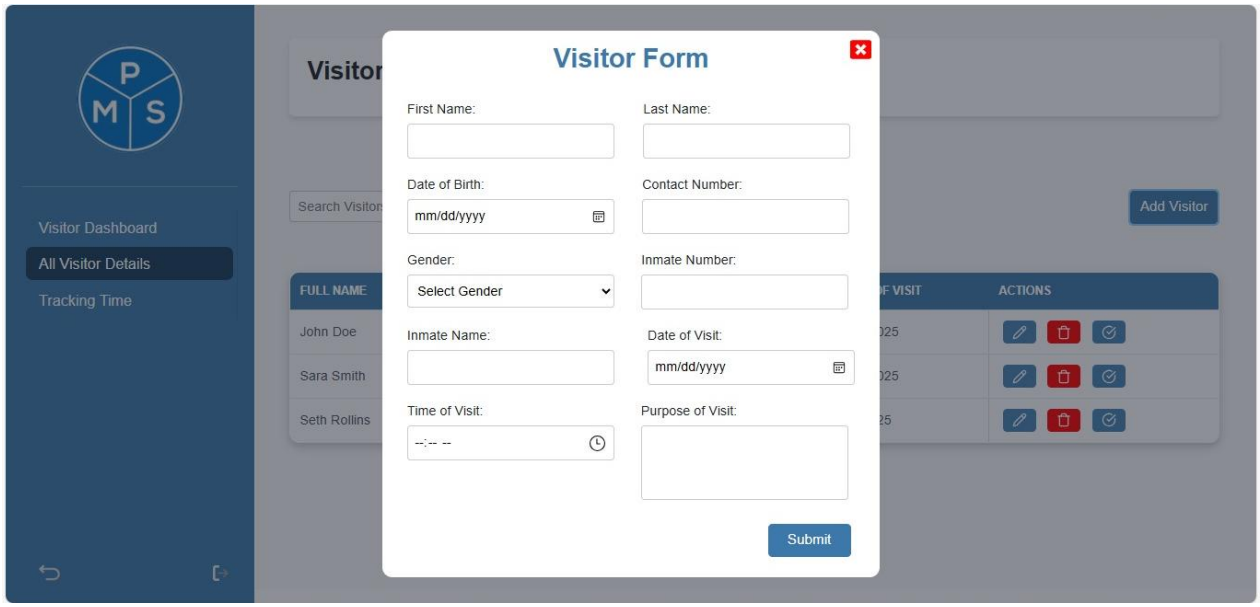

Doctors List
Staff Dashboard
All Jailors
All Doctors
Search
Add Doctor
Full Name
Medical License Number
Actions
John Doe
M200
Sara Smith
M400
Doctor Form
First Name:
Last Name:
Date of Birth:
mm/dd/yyyy
Contact Number:
Gender:
Male
Female
Specialty:
Medical License Number:
Start Date:
mm/dd/yyyy
Submit

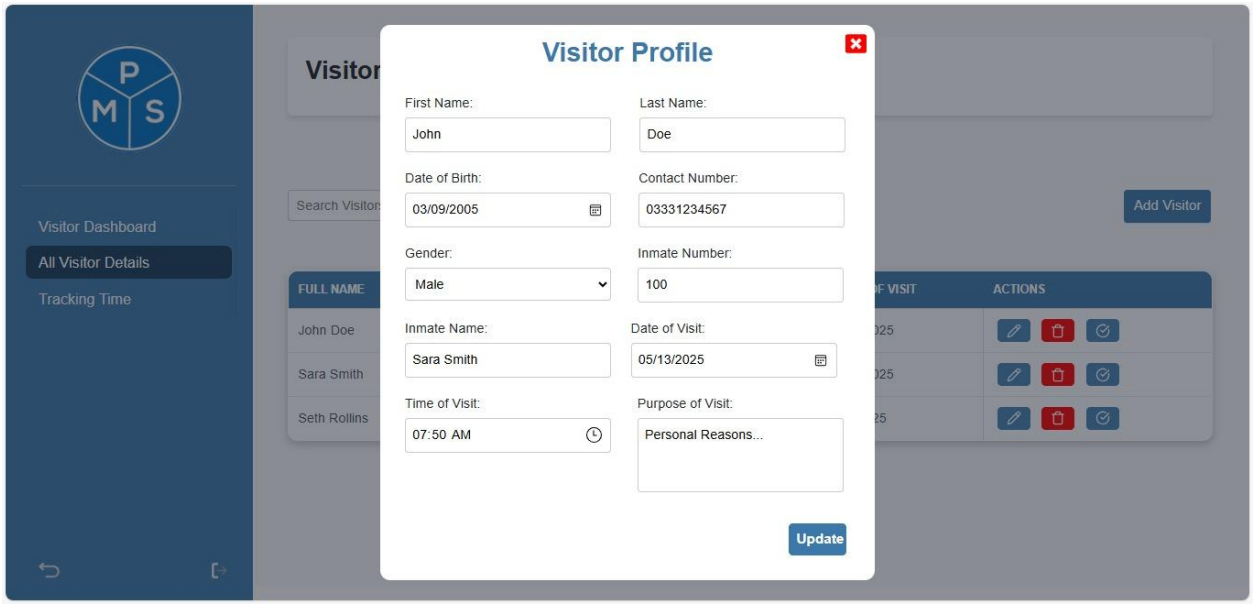

Doctors List
Staff Dashboard
All Jailors
All Doctors
Search
Add Doctor
Full Name
Medical License Number
Actions
John Doe
M200
Sara Smith
M400
Doctor Profile
First Name:
John
Last Name:
Doe
Date of Birth:
mm/dd/yyyy
Contact Number:
3331234567
Gender:
Male
Female
Specialty:
Heart Specialist
Medical License Number:
M200
Start Date:
mm/dd/yyyy
Update

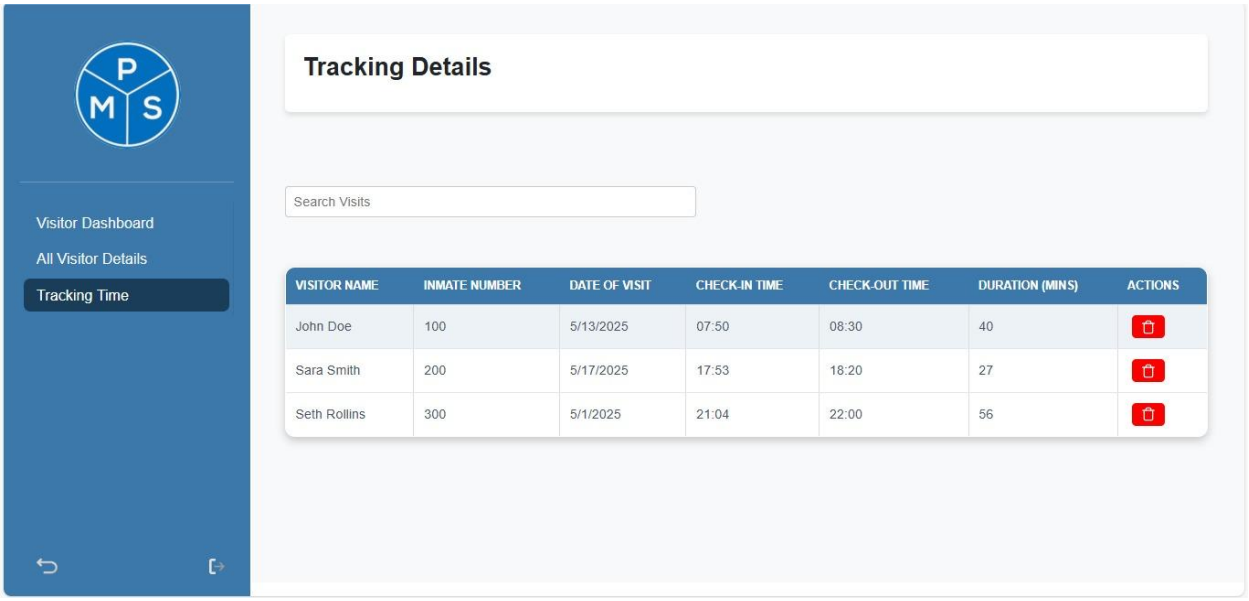
Visitor management



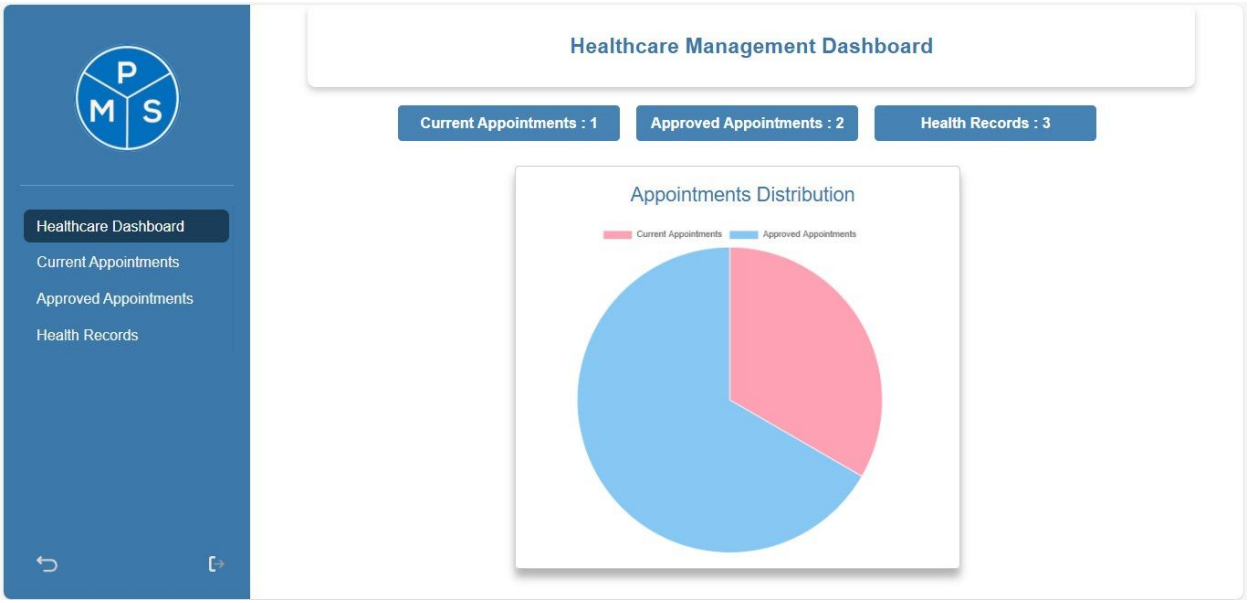


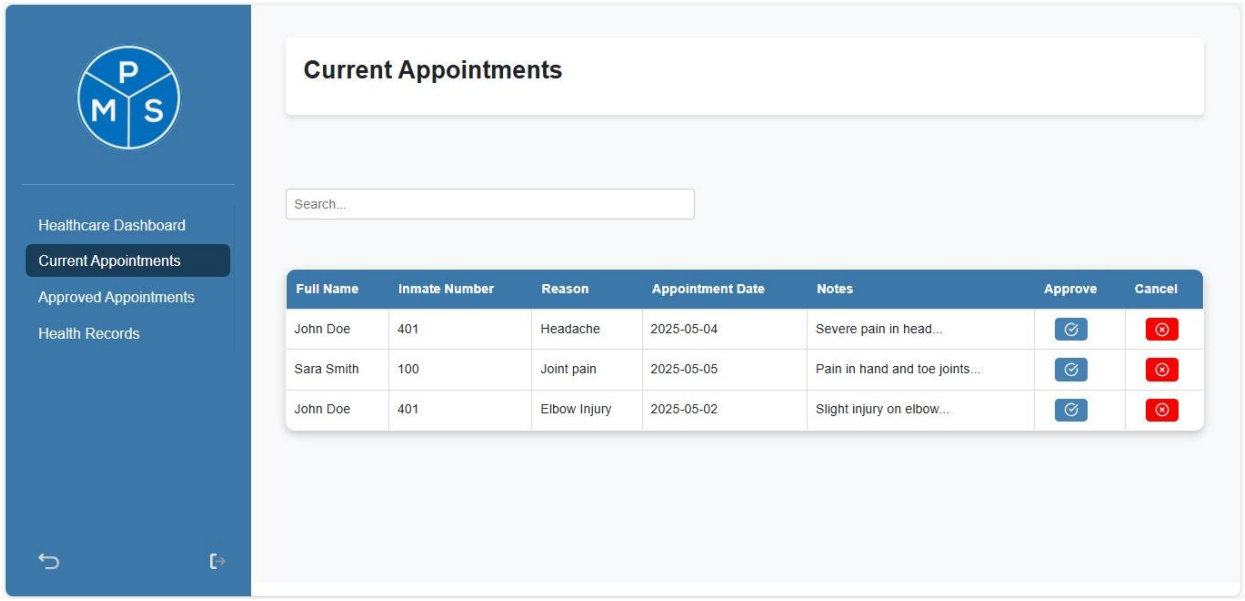

Visitor Form
First Name:
Last Name:
Date of Birth:
mm/dd/yyyy
Contact Number:
Gender:
Select Gender
Inmate Number:
Inmate Name:
Date of Visit:
mm/dd/yyyy
Time of Visit:
--:-- --
Purpose of Visit:
Submit
Visitor Dashboard
All Visitor Details
Tracking Time
Add Visitor
FULL NAME
ACTIONS
John Doe
Sara Smith
Seth Rollins

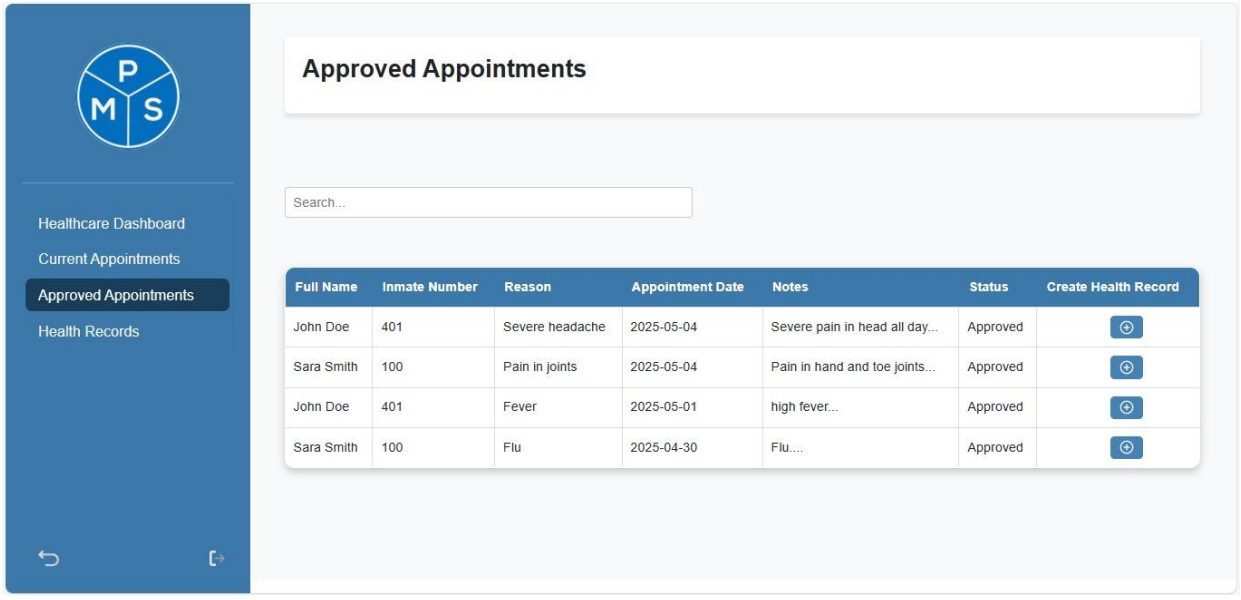

Visitor Profile
First Name:
John
Last Name:
Doe
Date of Birth:
03/09/2005
Contact Number:
03331234567
Gender:
Male
Inmate Number:
100
Inmate Name:
Sara Smith
Date of Visit:
05/13/2025
Time of Visit:
07:50 AM
Purpose of Visit:
Personal Reasons...
Update
Visitor Dashboard
All Visitor Details
Tracking Time
Add Visitor
FULL NAME
ACTIONS
John Doe
Sara Smith
Seth Rollins

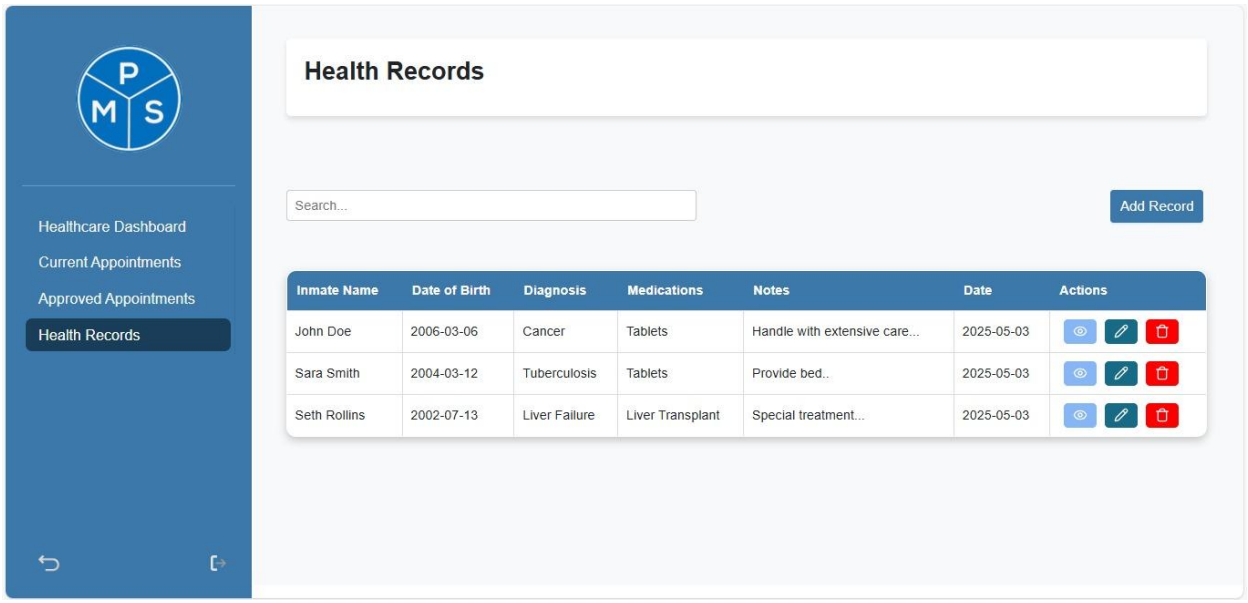


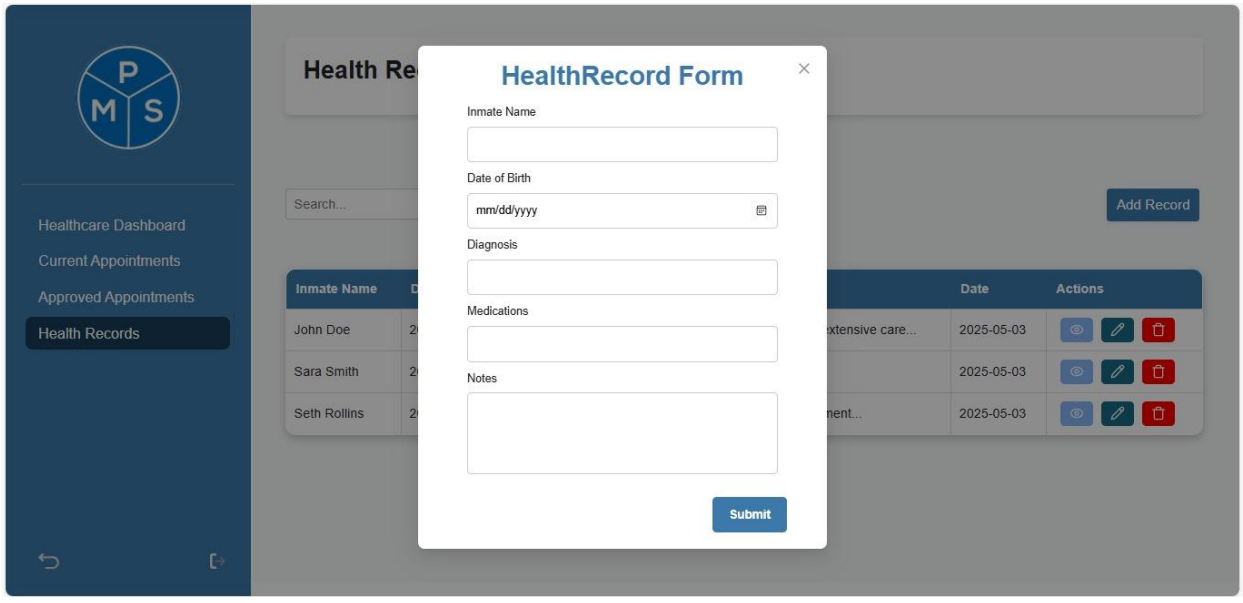
Health Management




Healthcare Dashboard
Current Appointments
Approved Appointments
Health Records
Current Appointments
Search...
Full Name
Inmate Number
Reason
Appointment Date
Notes
Approve
Cancel
John Doe
401
Headache
2025-05-04
Severe pain in head...
Sara Smith
100
Joint pain
2025-05-05
Pain in hand and toe joints...
John Doe
401
Elbow Injury
2025-05-02
Slight injury on elbow...


Healthcare Dashboard
Current Appointments
Approved Appointments
Health Records
Approved Appointments
Search...
Full Name
Inmate Number
Reason
Appointment Date
Notes
Status
Create Health Record
John Doe
401
Severe headache
2025-05-04
Severe pain in head all day...
Approved
Sara Smith
100
Pain in joints
2025-05-04
Pain in hand and toe joints...
Approved
John Doe
401
Fever
2025-05-01
high fever...
Approved
Sara Smith
100
Flu
2025-04-30
Flu....
Approved


Health Records
Search...
Add Record
Healthcare Dashboard
Current Appointments
Approved Appointments
Health Records
Inmate Name
Date of Birth
Diagnosis
Medications
Notes
Date
Actions
John Doe
2006-03-06
Cancer
Tablets
Handle with extensive care...
2025-05-03
Sara Smith
2004-03-12
Tuberculosis
Tablets
Provide bed..
2025-05-03
Seth Rollins
2002-07-13
Liver Failure
Liver Transplant
Special treatment...
2025-05-03


HealthRecord Form
Inmate Name
Date of Birth
mm/dd/yyyy
Diagnosis
Medications
Notes
Submit
Add Record
Healthcare Dashboard
Current Appointments
Approved Appointments
Health Records