Table of Contents

[Development Process Report for MediTech System 2](#_Toc166369211)

[Introduction 2](#_Toc166369212)

[System Architecture 2](#_Toc166369213)

[Database Design 2](#_Toc166369214)

[Feature Implementation 2](#_Toc166369215)

[Conclusion 3](#_Toc166369216)

[References: 3](#_Toc166369217)

[Task 1: Diagrams 4](#_Toc166369218)

[Use Case model 4](#_Toc166369219)

[Use Case Realisation (collaboration) Diagrams 5](#_Toc166369220)

[Sequence Diagram 6](#_Toc166369221)

[Class Model 7](#_Toc166369222)

[Activity Diagram model 8](#_Toc166369223)

[Python Meditech Application (Appendix) 9](#_Toc166369224)

[Login Window 9](#_Toc166369225)

[Main Menu & All Features 9](#_Toc166369226)

[Figure 1: MediTech Logo 2](file:///C:\Users\ammok\Desktop\MediTech-report.docx#_Toc166369548)

[**Figure 2:Use Case model** 4](#_Toc166369549)

[**Figure 3: MediTech- Use Case Actors Overview** 5](#_Toc166369550)

[**Figure 4: Patient Management – UML** 5](#_Toc166369551)

[**Figure 5:Appointment Management - UML** 5](#_Toc166369552)

[**Figure 6:Doctor Management – UML** 5](#_Toc166369553)

[**Figure 7: Sequence Diagram of appointment Management Use case** 6](#_Toc166369554)

[**Figure 8: Meditech Class Diagram** 7](#_Toc166369555)

[**Figure 9: Activity Diagram of Meditech** 8](#_Toc166369556)

[Figure 10: Login to System (with MongoDB User Credentials) 9](#_Toc166369557)

Medi Tech

# Development Process Report for MediTech System

## Introduction

A comprehensive healthcare management program with capabilities for patient administration, appointment scheduling, medical record keeping, billing, and report generating was the goal behind the creation of the MediTech System. Python was used in the development of the system, together with MongoDB for database administration and the Tkinter library for the graphical user interface (GUI). The main facets of the development process, such as database design, system architecture, and feature implementation, are described in this paper.

Figure : MediTech Logo

## System Architecture

The system follows a modular architecture to enhance maintainability and scalability. It consists of several modules, each responsible for a specific aspect of healthcare management. The main modules include Patient Management, Doctor Management, Staff Management, Appointment Management, Medical Records Management, Invoicing, Report Generation, and Payment Tracking.

The GUI is designed using Tkinter, providing a user-friendly interface with tabbed navigation for different functionalities. The system leverages a notebook widget to organize various tabs, enabling easy navigation between different modules.

## Database Design

The NoSQL database MongoDB was selected due of its scalability and flexibility. Patients' information was collected primarily for patient-related purposes, while bills' data was managed for invoicing purposes. Information like the patient's name, ID, contact details, and medical history are kept in the patient collection. Patient ID, services provided, total cost, and payment status are all tracked in the invoice collecting process. In order to link patients with their corresponding invoices, a linkage was formed between the patient and invoice collections. This design decision makes it possible to efficiently get patient-specific billing data.

## Feature Implementation

* **Login Authentication**

The Login Authentication feature ensures secure access to the system. Users are required to enter their credentials (username and password) to log in. The system verifies the entered credentials against those stored in the database. If the credentials are valid, the user is granted access to the system; otherwise, an error message is displayed. Also, after using application there is option to logout that will take the user to login screen again.

* **Patient Management**

Patient records may be added, changed, and removed using the Patient Management module. By including fields for pertinent medical data, the Tkinter GUI makes user interaction easier. To ensure durable data storage, patient records are kept in the MongoDB patient collection.

* **Doctor Management**

The creation, alteration, and deletion of doctor records is possible with the Doctor Management module, just like with Patient Management. The medical records module allows doctors to access patient records, which is a vital component of the healthcare administration process.

* **Staff Management**

The Staff Management module enables the addition and management of nursing and administrative staff. This module ensures that the healthcare facility's staffing needs are met.

* **Appointment Management**

Appointment scheduling is a critical feature of the system. The module allows for the creation of appointments, associating them with specific doctors and patients. This information is stored in the MongoDB appointments collection.

* **Medical Records Management**

The Medical Records Management module connects patients, doctors, and medical records. It allows for the creation and retrieval of medical records associated with specific patients and doctors.

* **Invoicing**

The Invoicing module automates the generation of invoices based on services rendered to patients. The user provides patient ID and a list of services, and the system calculates the total cost and generates an invoice. Invoices are stored in the MongoDB invoices collection.

* **Report Generation**

The Report Generation module creates sales and appointments reports. The reports are displayed in a new window, providing a summary of total costs for sales or a list of appointments.

* **Payment Tracking**

The Payment Tracking module allows users to check the payment status of an invoice by providing the invoice ID. It retrieves the payment status from the MongoDB invoices collection and displays the result.

## Conclusion

System architectural design, effective database administration with MongoDB, and thorough consideration of healthcare management requirements were all integral parts of the MediTech System development process. A pleasant user experience was ensured by the usage of Tkinter, which made it easier to create a graphical user interface (GUI). MongoDB's versatility and scalability made it an excellent option for managing data connected to healthcare.

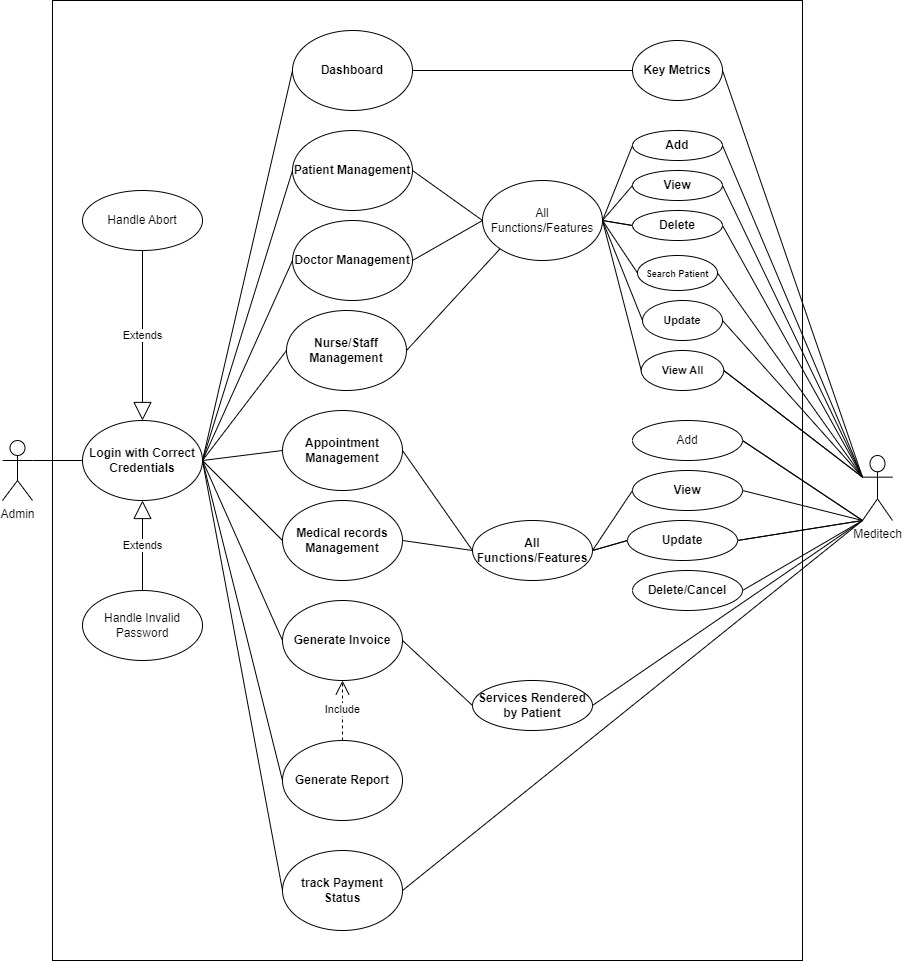
The system's modular architecture makes future expansion and maintenance simple. Because each module is made to carry out a certain set of tasks, code readability and reusability are encouraged. The program satisfies the prerequisites and offers a reliable healthcare administration solution.

## References:

* Python Documentation: <https://docs.python.org/3/>
* Tkinter Documentation: <https://docs.python.org/3/library/tkinter.html>
* MongoDB Documentation: <https://docs.mongodb.com/>
* Official MongoDB Python Driver Documentation: <https://pymongo.readthedocs.io/en/stable/>
* VS Code: <https://code.visualstudio.com/download>

Software Diagrams

Use Case model

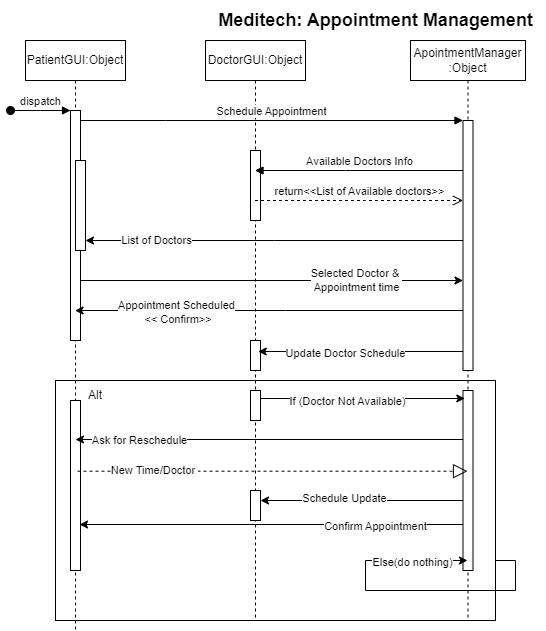


**Figure 2:Use Case model**

Use Case Realisation (collaboration) Diagrams

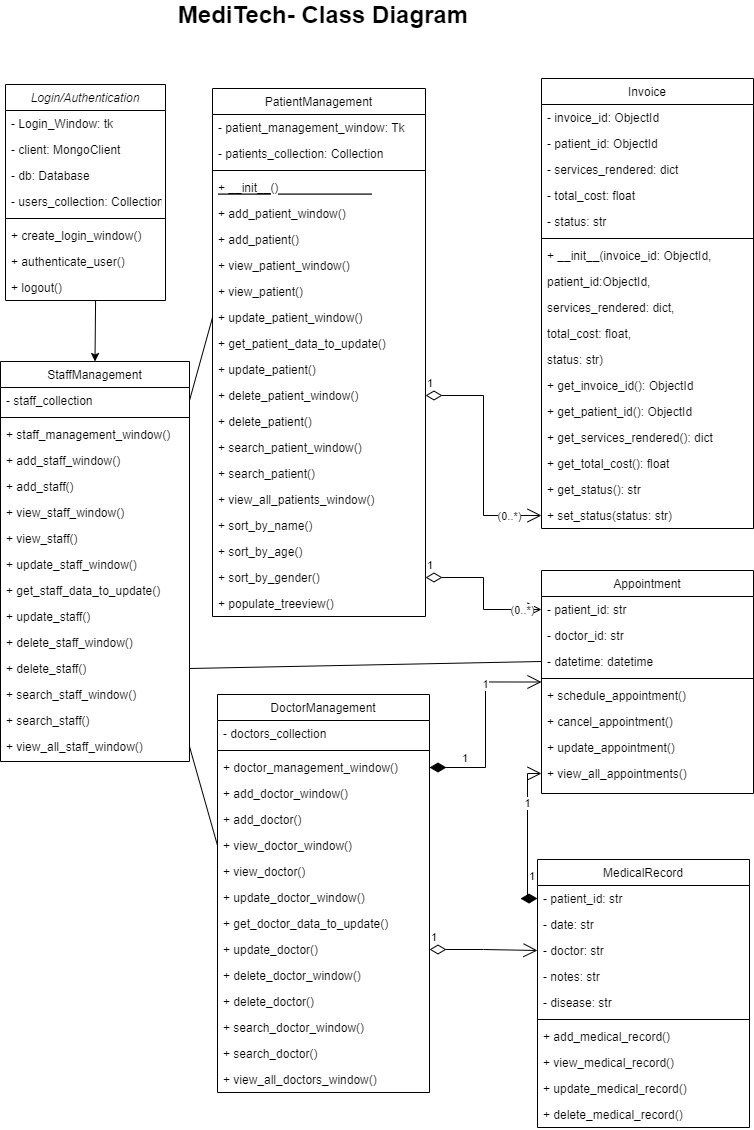
|  |  |
| --- | --- |
| **Figure 3: MediTech- Use Case Actors Overview** | **Figure 4: Patient Management – UML** |
| **Figure 5:Appointment Management - UML** | **Figure 6:Doctor Management – UML** |

Sequence Diagram



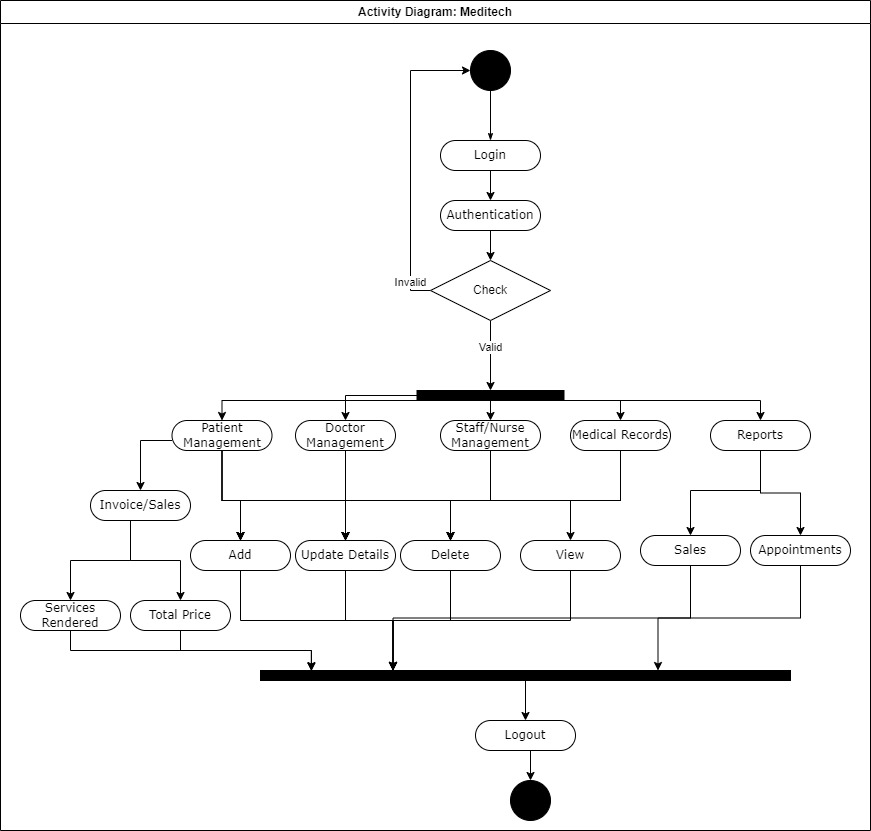
**Figure 7: Sequence Diagram of appointment Management Use case**

Class Model



**Figure 8: Meditech Class Diagram**

Activity Diagram model



**Figure 9: Activity Diagram of Meditech**

# Python Meditech Application (Appendix)

## Login Window

|  |
| --- |
| Figure 10: Login to System (with MongoDB User Credentials) |

## Main Menu & All Features

|  |  |  |
| --- | --- | --- |
|  | | |
|  |  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  |  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  |  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  |  | |
|  | |
|  | |
|  | |
|  |  | |
|  | |
|  | |
|  | |
|  | |  |
|  | |  |
|  | |  |