Introduction:

In System Analysis and Design, Object-Oriented Analysis and Design (OOAD) is a systematic approach used to analyze, design, and build systems based on real-world objects and their interactions. It helps break down complex systems into manageable components by modeling them as objects with attributes and behaviors. As part of the object-oriented approach, the Unified Modeling Language (UML) was introduced as a standardized way to visualize system designs. UML enables developers, analysts, and stakeholders to communicate ideas clearly through a set of diagrams that capture both the structure and behavior of a system. Among the various UML diagrams, the Use Case Diagram is particularly significant during the early stages of analysis and design. It captures a system's functional requirements by showing how users (actors) interact with different services (use cases). This ensures that all stakeholder needs are considered during system development.

In our system, CUET Medical Center, the Use Case Diagrams illustrate how various users such as patients, doctors, and medical personnel interact with the system for services such as diagnostics, stock tracking, and reporting. By clearly outlining these interactions, the Use Case Diagram provides a structured overview of the system's scope and supports effective system design and development.

UML Diagrams:

The Unified Modeling Language (UML) is a standardized modeling language used to visualize, specify, construct, and document the components of a system. UML helps developers, designers, and stakeholders understand and communicate the structure and behavior of a system before actual implementation.

Categories of UML Diagrams:

UML diagrams are broadly divided into two main types:

- Structural Diagrams
- Behavioral Diagrams

Each serves a different purpose, but together they provide a comprehensive view of the system.

Structural Diagrams (Static View):

Structural diagrams focus on the static aspects of a system, describing what the system consists of.

- **Class Diagram:** The most common UML diagram, it models the classes within a system, their attributes (data members), methods (functions), and the relationships (such as inheritance, association, aggregation, and composition) between classes. It serves as the backbone of object-oriented design.
- **Object Diagram:** Represents specific instances of classes (objects) at a particular point in time, showing the relationships between objects during execution.
- **Composite Structure Diagram:** Details the internal structure of a class and the collaborations between its parts.
- **Deployment Diagram:** Describes the physical deployment of software artifacts onto hardware nodes, showing the system's physical architecture.
- **Package Diagram:** Organizes classes and other elements into packages and shows dependencies between these higher-level groupings.

Behavioral Diagrams (Dynamic View):

Behavioral diagrams describe the dynamic behavior of the system, focusing on how it responds to events over time.

- **Use Case Diagram:** Represents the system's functional requirements by depicting actors (users or external systems) and the use cases (specific functionalities) they interact with. It helps identify the system's scope and main functions from an end-user perspective.
- **Sequence Diagram:** Shows object interactions arranged in time sequence, highlighting the flow of messages between objects to accomplish specific behaviors.
- **Activity Diagram:** Models workflows and business processes, illustrating the flow from one activity to another, including decision points, parallel processes, and concurrency.
- **State Machine Diagram:** Describes the different states an object or system can be in and how it transitions from one state to another based on events.
- **Communication Diagram:** Similar to sequence diagrams but emphasizes the relationships between objects and the messages they exchange.
- **Timing Diagram:** Focuses on changes in the state or condition of an instance over time.

UML Diagram:

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. It helps in designing and characterizing, especially those software systems that incorporate the concept of Object orientation. It describes the working of both the software and hardware systems. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. Basic building block of UML-

- Things
- Relationship
- Diagrams

Use Case Diagram:

A use case diagram is used to represent the functional requirements of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system. It provides a way for developers, domain experts and end-users to Communicate. Use Case describes the functional behavior of the system from the user's point of view. Used during the requirements elicitation stage to represent external behavior. It has-

- **Actor**: An actor represents a user, external system, or physical environment that interacts with the system. Actors initiate use cases to achieve specific goals but are not part of the system itself.

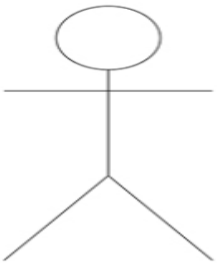

Figure 1: Actor

- **Use Case:** A use case represents a specific function or behavior the system provides in response to an actor's interaction. It defines a complete flow of events triggered by the actor to achieve a goal.

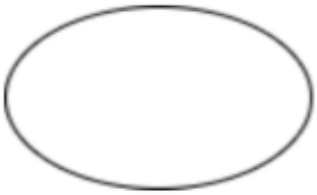

Figure 2: Use Case

- **Boundary Box:** The boundary box defines the scope of the system. It encloses all use cases and separates them from external actors. Anything inside the box is part of the system; anything outside (like actors) interacts with it.

Figure 3: Boundary Box

- **Relationships:** It is used to describe relationships between actors and use cases.
  1. **Association**: Shows interaction between an actor and a use case using a solid line. It represents that the actor is involved in the execution of the use case.

Figure 4: Association

  2. **Include (<<include>>)**: Represents mandatory behavior reused by multiple use cases. Helps avoid duplication. Shown with a dashed arrow labeled <<include>>.

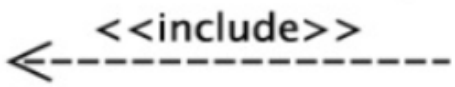

Figure 5: Include

  3. **Extend (<<extend>>)**: Represents optional or conditional behavior that extends a base use case. Shown with a dashed arrow labeled <<extend>>.

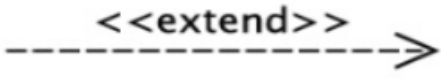

Figure 6: Extend

4. **Generalization**: Indicates inheritance between actors or use cases. A child actor/use case inherits behavior from a parent. Represented by a solid line with a hollow triangle pointing to the parent.

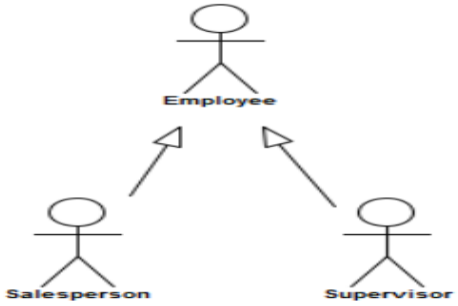

Figure 7: Generalization

1. Use Case Diagram for Student Health Portal:

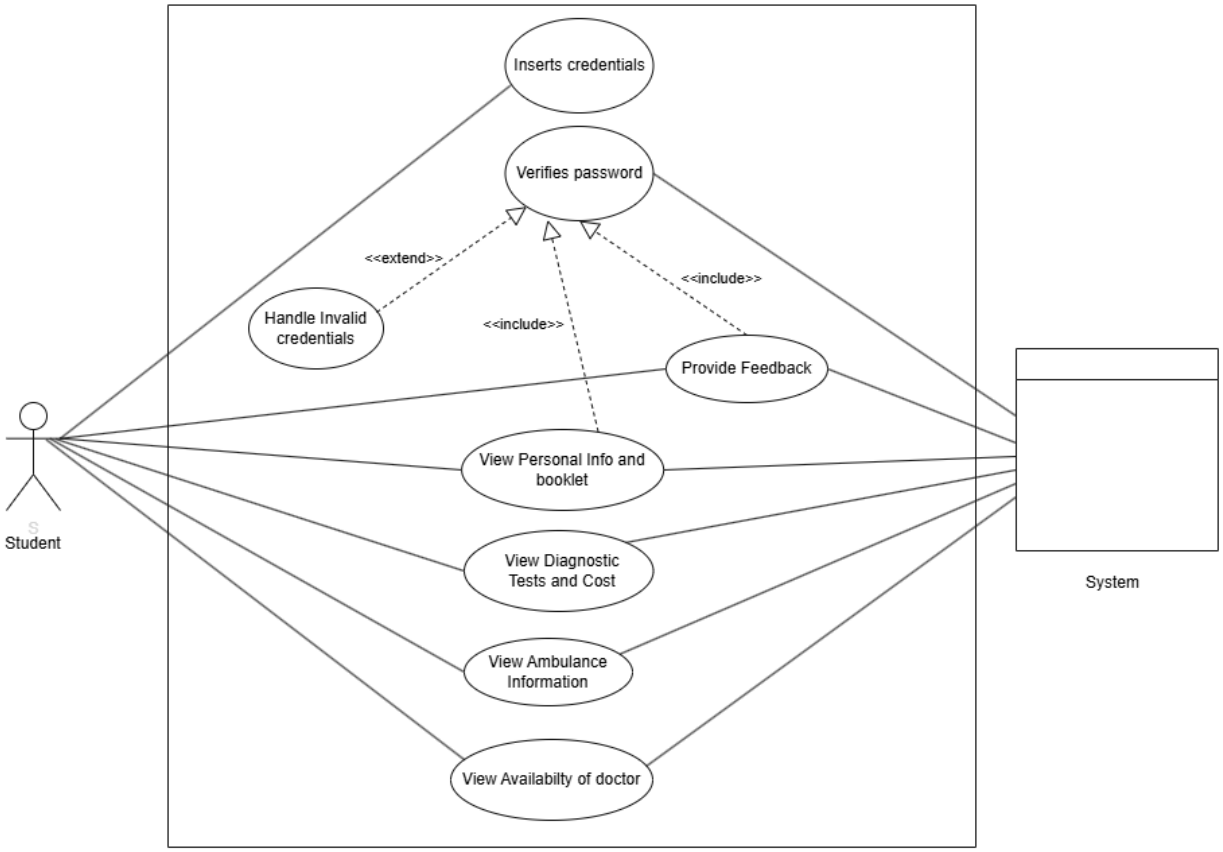

Figure-8: Use-Case Diagram for student health information access system

Description:

The website for the CUET Medical Center is designed primarily to serve students as the main actor, providing them with access to their personal medical information and relevant medical center data. Additionally, the Database acts as a system actor responsible for storing and supplying all necessary data to the website.

Actors:

Student: The primary user who logs into the website to view and interact with their medical information and related services.

System:

The core data management service that stores all patient records, operational data, medical histories, doctor availability, ambulance information, and feedback entries.

Use Cases:

- **Insert Credentials:** The student initiates the login process by entering their username and password into the system's authentication interface.
- **Verify Password:** The system checks the entered credentials against the records stored in the database. Successful verification grants the student access to the platform.
- **Handle Invalid Credentials:** This use case is activated when incorrect login credentials are submitted. The system displays an error message and prompts the user to try again.
- **View Personal Information and Medical Booklet:** After successful login, the student can view their digitized medical records, which include previous appointments, diagnoses, prescriptions, and health history.
- **View Doctor Availability:** Provides the student with up-to-date information on which doctors are available at the medical center, along with their consultation schedules.
- **View Diagnostic Tests and Associated Costs:** Displays a comprehensive list of diagnostic tests offered by the medical center and their respective costs, enabling students to make informed decisions.
- **View Ambulance Information:** Provides real-time access to ambulance availability and contact details of the on-duty ambulance drivers.
- **Provide Feedback:** Allows students to submit feedback or suggestions about the medical services they have received. Feedback is stored in the system for review and analysis.

Use Case Relationships:

- **<<include>> Relationship:** The <<include>> relationship is used to represent a mandatory dependency between use cases, where one use case always incorporates the behavior of another. In CUET Medical Center system, two <<include>> relationships are defined. The use case "View Personal Info and Booklet" includes "Verifies Password", ensuring that a student must be authenticated before accessing sensitive medical data. Additionally, the use case "Verifies Password" includes "Provide Feedback", integrating the feedback submission process as part of the post-login interaction. These inclusions promote functional reuse and help maintain the integrity and consistency of user interactions across the system.
- **<<extend>> Relationship:** The <<extend>> relationship denotes an optional or conditionally executed behavior that occurs only under specific circumstances. In the CUET Medical Center system, the use case "Handle Invalid Credentials" is defined as an extension of "Verifies Password". This relationship reflects the scenario in which the system invokes the error-handling process only when incorrect credentials are submitted during login.

2. Use Case Diagram for CUET Medical Center Desktop App:

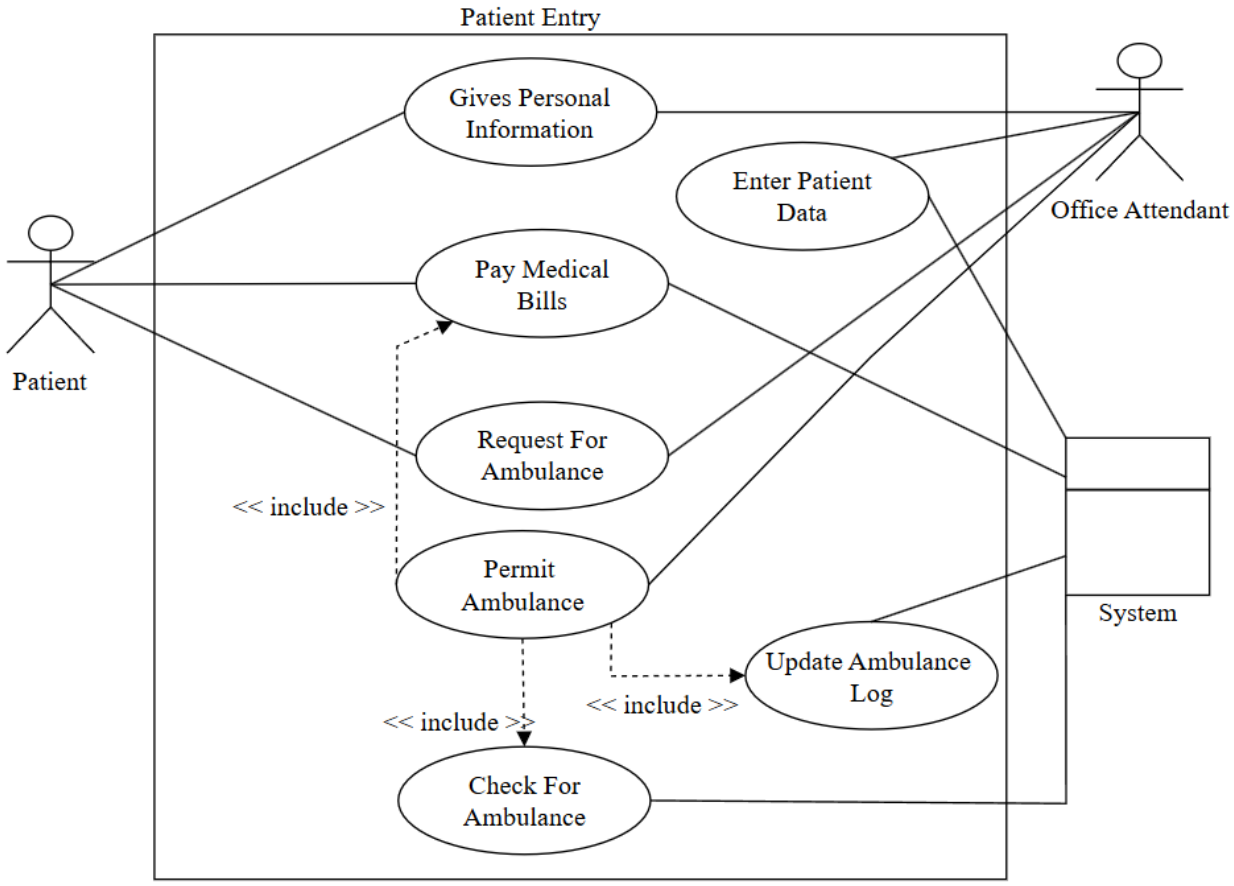

Figure-9: Use Case Diagram of Patient Entry

Description:

Actors:

- Patient.
- Office Attendant.
- System.

Use Cases:

- **Gives Personal Information:** The patient provides their personal information to office attendant.
- **Pay Medical Bills:** The patient completes payment for medical services received**.**
- **Enter Patient Data:** The office attendant inputs the patient's details into the system.
- **Request for Ambulance:** The patient requests ambulance services to the office attendant.

- **Permit Ambulance:** The office attendant authorizes the ambulance request after verification.
- **Update Ambulance Log:** The system updates ambulance logs.
- **Check for Ambulance:** The system helps to check the availability of ambulance.

Use Case Relationships:

- **Request For Ambulance <<include>> Pay Medical Bills:** Ambulance request process requires the patient to have paid the medical bills.
- **Permit Ambulance <<include>> Check for Ambulance**: Ambulance permission involves verifying ambulance availability.
- **Permit Ambulance <<include>> Update Ambulance Log:** Granting ambulance permission triggers the system to update ambulance records.

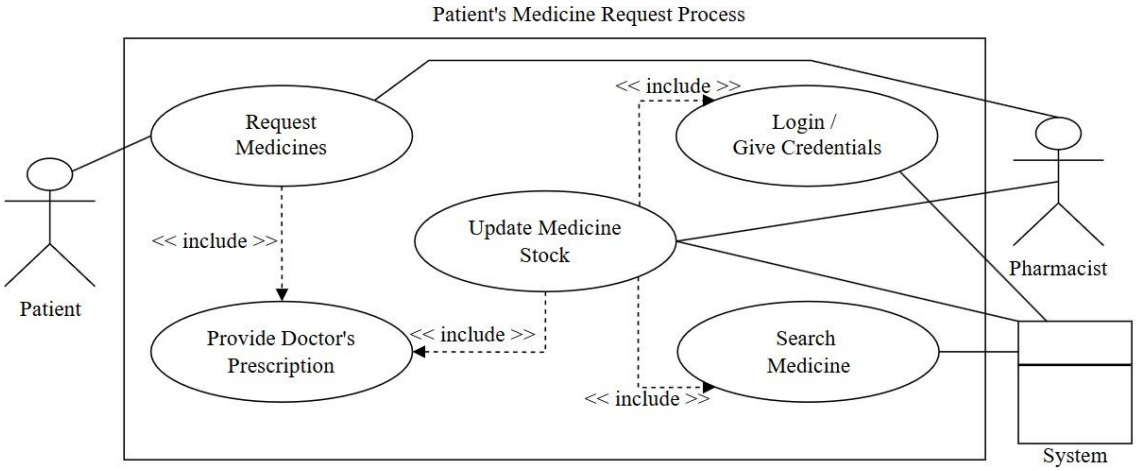

Figure-10: Use Case Diagram of Patient's Medicine Request

Description:

Actors:

- Patient.
- Pharmacist.
- System.

Use Cases:

- **Request Medicines**: The patient requests medicines from the pharmacist.
- **Provide Doctor's Prescription**: The patient provides a valid doctor's prescription for the requested medicines.

- **Update Medicine Stock:** The pharmacist updates the medicine stock in the system after dispensing.
- **Login / Give Credentials**: The pharmacist logs into the system or provides credentials to access the system.
- **Search Medicine:** The pharmacist searches for medicines in the system database.

Use Case Relationships:

- **Request Medicines <<include>> Provide Doctor's Prescription:** Medicine requests must be accompanied by a doctor’s prescription.
- **Update Medicine Stock <<include>> Provide Doctor's Prescription:** Updating stock involves the prescription being provided.
- **Update Medicine Stock <<include>> Login / Give Credentials:** Pharmacist must be logged in to update medicine stock.
- **Update Medicine Stock <<include>> Search Medicine**: Pharmacist searches the system to verify medicine availability before updating stock.

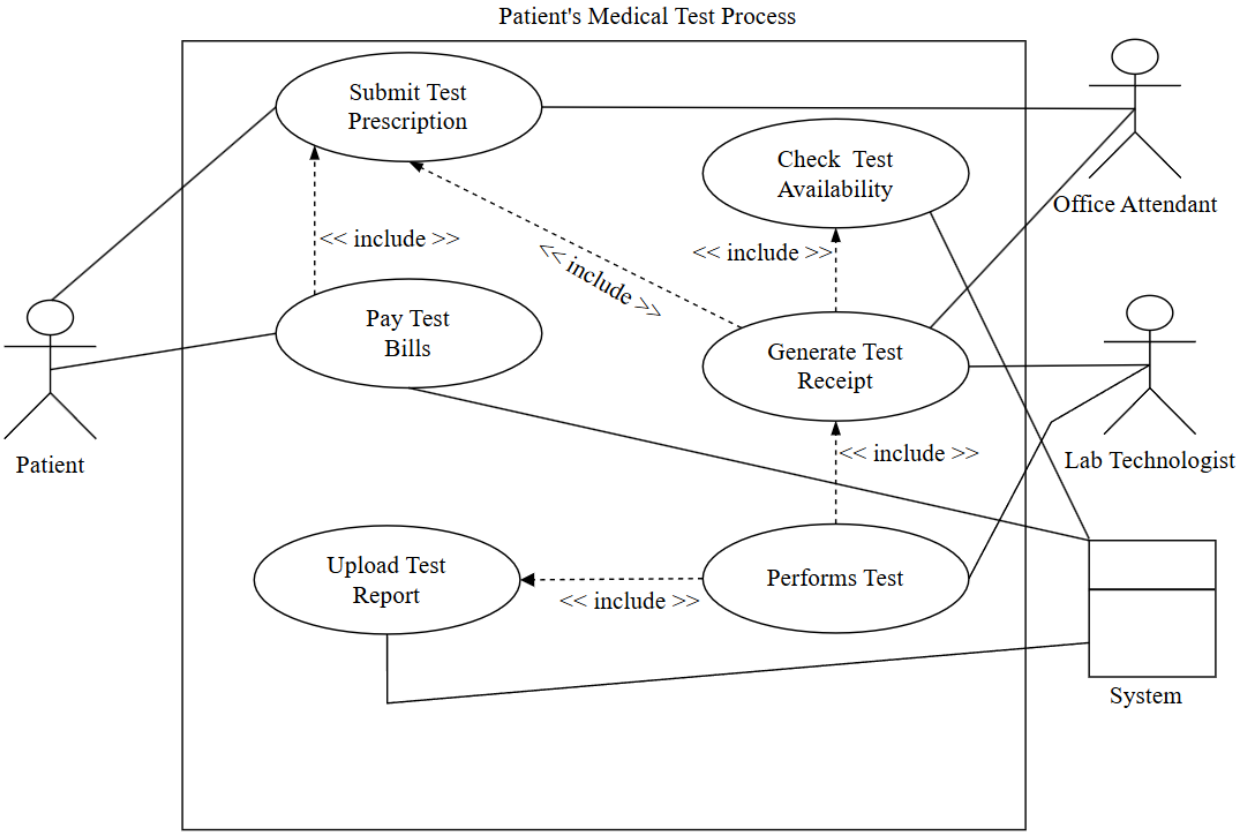

Figure-11: Use Case Diagram of Patient's Medical Test Process

Description:

Actors:

- Patient.
- Office Attendant.
- Lab Technologist.
- System

Use Cases:

- **Submit Test Prescription:** The patient submits the test prescription to the office attendant.
- **Pay Test Bills:** The patient pays for the medical tests.
- **Check Test Availability:** The office attendant verifies if the requested test is available.
- **Generate Test Receipt:** The office attendant and lab technologist generate receipts.
- **Performs Test:** The lab technologist performs the medical test on the patient.
- **Upload Test Report:** The lab technologist uploads the test reports to the system.

Use Case Relationships:

- **Pay Test Bills <<include>> Submit Test Prescription:** Submitting a test prescription requires payment of the test fees.
- **Check Test Availability <<include>> Submit Test Prescription:** Availability of the test is checked during the prescription submission.
- **Generate Test Receipt <<include>> Submit Test Prescription:** A receipt is generated as part of the prescription process.
- **Performs Test <<include>> Generate Test Receipt:** Performing the test requires a receipt to be generated first.
- **Performs Test <<include>> Upload Test Report:** Test performance concludes with uploading the test report to the system.

Conclusion:

The use case diagrams for the CUET Medical Center system provide a clear and comprehensive overview of the interactions between users and the system across multiple healthcare processes. By identifying key actor such as patients, office attendants, pharmacists, and lab technologists, and outlining their specific roles, these diagrams effectively capture the functional requirements of the system. The inclusion of system components highlights the essential role of databases in managing patient data, medical billing, medicine stock, and test reports.

Moreover, the use case relationships, especially the <<include>> dependencies, demonstrate the logical flow and dependencies between different operations, ensuring that critical prerequisites such as authentication, payment, and availability checks are systematically enforced. This structured approach facilitates efficient system design, development, and communication among stakeholders, ultimately supporting the delivery of reliable, user-friendly medical services. Overall, the use case models lay a strong foundation for building a robust and well-organized medical management system that addresses both administrative and clinical needs.