Index

Sr. No.	Name of Experiment	Page No.	Date of Experiment	Date of Submission	Signature
1	Prepare a SRS document in line with the IEEE recommended standards.		17/02/2025	24/02/2025	
2	Draw the use case diagram and specify the role of each of the actors. Also state the precondition, post condition and function of each use case.		24/02/2025	03/03/2025	
3	Draw the activity diagram.		03/03/2025	10/03/2025	
4	Identify the classes. Classify them as weak and strong classes and draw the class diagram.		10/03/2025	17/03/2025	
5	Draw the sequence diagram for any two scenarios.		17/03/2025	04/04/2025	
6	Draw the collaboration diagram.		17/03/2025	04/04/2025	
7	Draw the state chart diagram.		04/04/2025	11/04/2025	
8	Draw the component diagram.		04/04/2025	11/04/2025	
9	Perform forward engineering in java. (Model to code conversion)		11/04/2025	25/04/2025	
10	Perform reverse engineering in java. (Code to Model conversion)		11/04/2025	25/04/2025	
11	Draw the deployment diagram.		25/04/2025	09/05/2025	

Experiment-10

Aim: Perform reverse engineering (Code to Model conversion)

Reverse Engineering: Code to Model Conversion

Introduction

In the ever-evolving world of software development, maintaining and understanding existing systems is as important as creating new ones. As systems grow in size and complexity, it becomes increasingly difficult to manage them without a clear understanding of their internal workings. This is where **reverse engineering** comes into play. Reverse engineering refers to the process of analyzing a system to identify the system's components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction.

A significant aspect of reverse engineering in software engineering is **Code to Model conversion**, where the existing source code is examined to reconstruct higher-level design models such as UML (Unified Modeling Language) diagrams. This approach helps developers and stakeholders understand the structure, behavior, and design of an existing software system without having direct access to the original design documentation.

Definition and Purpose

Reverse engineering, specifically in the context of software, is the process of extracting knowledge or design information from existing code. It involves analyzing the code to reconstruct useful models that reflect the software's structure and behavior. The purpose of Code to Model conversion is to translate low-level source code into meaningful, high-level design artifacts that aid in:

- **Understanding legacy systems**: Many older systems lack proper documentation, and reverse engineering provides insights into their inner workings.
- **Facilitating software maintenance**: Developers can better maintain systems when they understand their architecture and design.
- **Ensuring compliance**: Reverse engineering helps verify that the implementation matches the intended architecture or design specifications.
- **Supporting migration**: When migrating software to new platforms or languages, reverse engineering simplifies the process by clarifying system components.
- **Aiding in documentation**: Automatically or semi-automatically generated models serve as updated documentation for the system.

The Process of Code to Model Conversion

The reverse engineering process involves multiple steps that collectively help in generating useful models from the source code. These steps include:

Code Parsing and Analysis

This step involves reading and parsing the source code to gather syntactic and semantic information. This includes identifying classes, methods, variables, data types, dependencies, and relationships between components. Lexical analysis tools or parsers are often used to break down the code into manageable components.

Abstract Model Generation

Once the code has been analyzed, the next step is to create abstract models that reflect the architecture and behavior of the system. These models typically include:

-
    - **Class Diagrams**: Represent classes, their attributes, operations (methods), and relationships (e.g., inheritance, association, composition).
    - **Object Diagrams**: Depict instances of classes at a particular point in time.
    - **Sequence Diagrams**: Describe the flow of messages between objects to carry out a function.
    - **Activity Diagrams**: Visualize workflows and the activities involved in a process.
    - **Use Case Diagrams**: Represent the interaction between users (actors) and the system.

Tool Support

A variety of tools are available that automate parts of the reverse engineering process. These tools can analyze source code and generate UML diagrams. Some popular tools include:

-
    - **StarUML**
    - **Visual Paradigm**
    - **Enterprise Architect**
    - **Eclipse Modeling Framework (EMF)**
    - **ArgoUML**

These tools improve accuracy and save time by generating consistent and standardized models from source code.

Manual Refinement

The models generated through tools often contain redundant or low-level details that may not be useful at the design level. Manual refinement involves simplifying these models, removing unnecessary components, and restructuring the diagrams for better readability and comprehension.

Advantages of Code to Model Conversion

Code to Model conversion offers several benefits for software engineers, project managers, and system architects:

1. **Improved Comprehension**: Models help in visualizing the system architecture, which is especially useful for onboarding new developers.
2. **Documentation Recovery**: Generates design documentation from existing systems, which is often missing or outdated.
3. **System Reusability**: Identifies reusable components or modules that can be extracted for use in other projects.
4. **Enhanced Maintenance**: Facilitates understanding of legacy code, making it easier to troubleshoot and update

Challenges in Reverse Engineering

While reverse engineering has many advantages, it also presents several challenges:

1. **Complexity of Source Code**: Large and poorly structured codebases are difficult to analyze and model accurately.
2. **Lack of Comments and Documentation**: Absence of meaningful comments makes it harder to understand the intent behind code sections.
3. **Dynamic Behavior**: Features like dynamic binding, reflection, or runtime polymorphism are hard to represent in static models.
4. **Tool Limitations**: Tools may not support all programming languages or code constructs, leading to incomplete models.
5. **Time Consumption**: The process can be time-intensive, especially when models need significant manual refinement.

Applications of Reverse Engineering

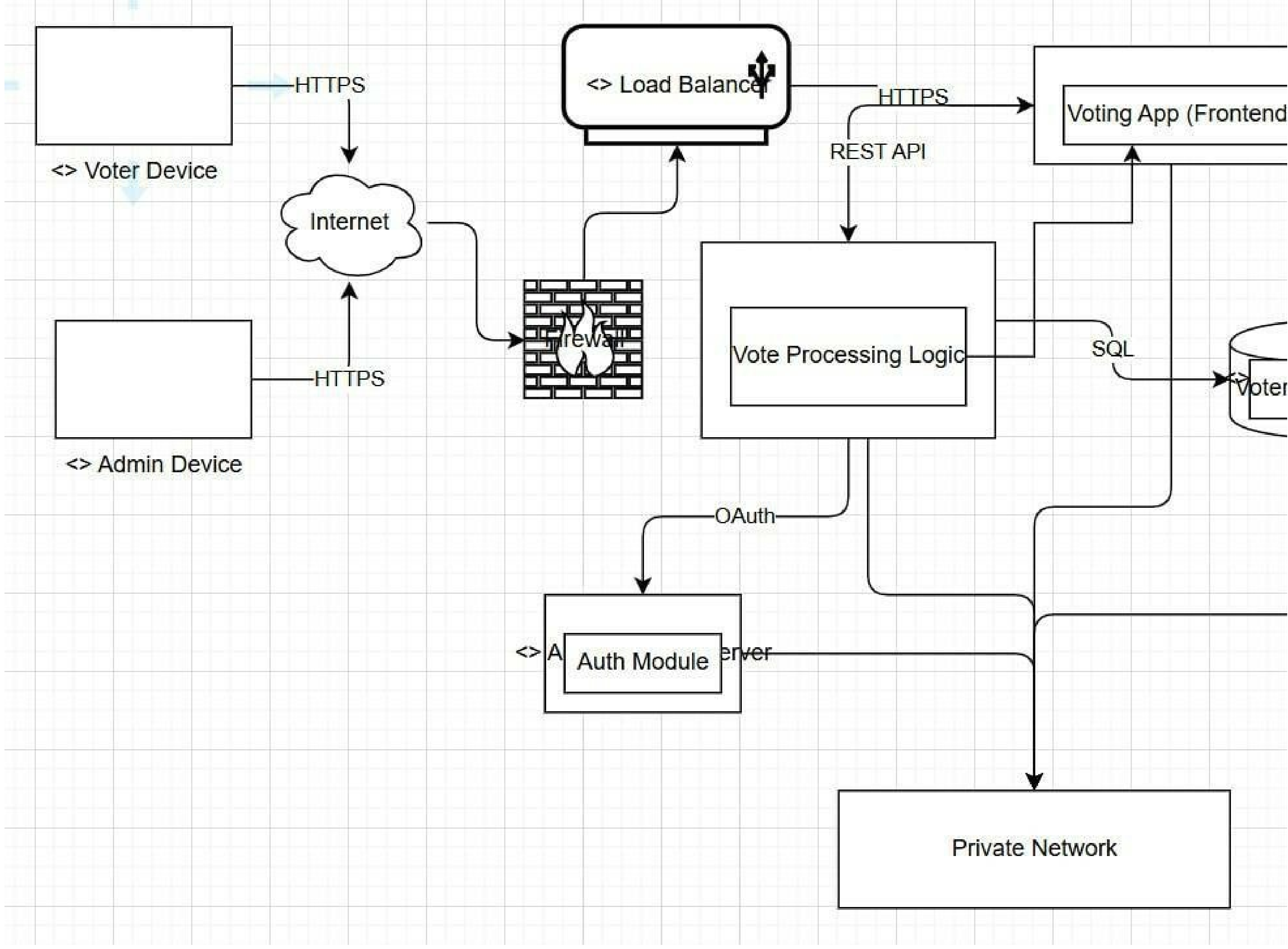
Reverse engineering through code-to-model conversion finds applications in multiple areas:

-
    - **Software Maintenance and Evolution**: Understanding systems for adding new features or fixing bugs.
    - **Security Analysis**: Inspecting binary or source code for vulnerabilities and threats.
    - **Legacy System Modernization**: Migrating old software to new platforms or languages.
    - **Training and Education**: Teaching software design principles using real-world code examples.

Experiment-11

Aim: Draw the deployment diagram.

Deployment diagram of Online Voting System



Deployment diagram of Online Voting System

A **deployment diagram** in the Unified Modeling Language (UML) is used to model the physical architecture of a system, illustrating how software and hardware components are deployed and interact within a system. It provides a high-level view of the system's physical structure, showing the distribution of software artifacts across hardware nodes and the communication pathways between them.

Key Concepts of Deployment Diagrams

Purpose:

-
  1. Deployment diagrams depict the physical deployment of software artifacts on hardware nodes, focusing on the system's runtime architecture.
  2. They help stakeholders understand how system components are distributed, how they communicate, and how they utilize hardware resources.
  3. They are particularly useful for system architects, developers, and administrators to plan and manage the deployment of software systems.

Components:

-
  1. **Nodes**: Represent physical hardware devices or execution environments (e.g., servers, devices, or virtual machines). In the provided diagram, nodes include "Voter Device," "Admin Device," "Web Server," "Application Server," "Database Server," "Authentication Server," and "Load Balancer."
  2. **Artifacts**: Represent software components or files deployed on nodes (e.g., applications, libraries, or data files). In the diagram, artifacts include "Voting App (Frontend)," "Vote Processing Logic," "Voter & Vote Data," and "Auth Module."
  3. **Communication Paths**: Represent connections between nodes, often labeled with protocols or technologies (e.g., HTTPS, REST API, SQL, OAuth in the diagram).
  4. **Stereotypes**: Used to categorize nodes or artifacts (e.g., <<device>>, <<server>>, <<database>>, <<load balancer>> in the diagram), providing clarity on their roles.

Key Elements in the Diagram:

-
  1. **Voter and Admin Devices**: Represent client devices accessing the system, likely via browsers or apps, communicating over HTTPS to the internet.
  2. **Internet Cloud**: Acts as an intermediary, showing external network connectivity.

3. **Firewall**: Ensures security by filtering incoming and outgoing traffic.
4. **Load Balancer**: Distributes incoming traffic across servers to optimize performance and reliability.
5. **Web Server**: Hosts the "Voting App (Frontend)," serving the user interface.
6. **Application Server**: Runs the "Vote Processing Logic," handling business logic and communicating with the web server via REST API.
7. **Database Server**: Stores "Voter & Vote Data," accessed via SQL queries from the application server.
8. **Authentication Server**: Manages user authentication using the "Auth Module," integrated via OAuth.
9. **Private Network**: Connects internal servers, indicating a secure, isolated network for backend communication.

Notations:

-
    1. Nodes are typically represented as 3D boxes (e.g., servers) or specialized shapes (e.g., cylinders for databases, clouds for networks).
    2. Artifacts are depicted as rectangles with a folded top-right corner, often placed inside nodes to show deployment.
    3. Communication paths are shown as lines, often labeled with protocols or technologies to indicate the type of interaction.

Theoretical Aspects of Deployment Diagrams

System Scalability:

-
    1. The inclusion of a load balancer in the diagram indicates a design for scalability, allowing the system to handle multiple user requests by distributing them across web servers.
    2. Deployment diagrams help identify bottlenecks and plan for horizontal scaling (adding more nodes) or vertical scaling (upgrading node resources).

Security Considerations:

-
    1. The firewall and authentication server highlight security mechanisms. The diagram shows secure communication (HTTPS, OAuth) and a private network to protect sensitive data.
    2. Deployment diagrams aid in mapping security components and ensuring compliance with data protection standards.

Distributed Systems:

-
    1. The diagram represents a distributed architecture, with separate servers for web, application, database, and authentication functions. This separation enhances modularity and fault tolerance.
    2. Deployment diagrams are critical for distributed systems, as they clarify how components are physically or logically separated.

Communication Protocols:

-
    1. The use of specific protocols (HTTPS, REST API, SQL, OAuth) in the diagram specifies how nodes interact, which is essential for ensuring interoperability and performance.
    2. Deployment diagrams help developers choose appropriate protocols and technologies for communication.

Deployment Planning:

-
    1. The diagram provides a blueprint for deploying the system, showing which artifacts reside on which nodes and how nodes are interconnected.
    2. It aids in resource allocation, network configuration, and system maintenance.

Real-World Applications:

-
    1. Deployment diagrams are used in various domains, such as web applications, enterprise systems, and cloud- based solutions. In this case, the diagram models an online voting system, emphasizing secure and scalable deployment.
    2. They are valuable during the design, implementation, and maintenance phases of software development.

Benefits of Deployment Diagrams

- **Clarity**: Provide a clear visual representation of the system's physical architecture, making it easier for technical and non-technical stakeholders to understand.
- **Planning**: Assist in planning hardware requirements, network configurations, and software deployment strategies.
- **Troubleshooting**: Help identify issues related to deployment, such as misconfigured networks or overloaded servers.
- **Documentation**: Serve as documentation for the system's architecture, aiding future maintenance and upgrades.

Limitations

- **Static Nature**: Deployment diagrams typically represent a snapshot of the system at a specific time, which may not account for dynamic changes (e.g., auto-scaling in cloud environments).
- **Complexity**: For large systems, diagrams can become cluttered, requiring careful organization to remain readable.
- **Limited Behavioral Insight**: They focus on physical deployment rather than the system's runtime behavior or logic, which is better addressed by other UML diagrams (e.g., sequence or activity diagrams).

GALGOTIAS COLLEGE OF ENGINEERING AND TECHNOLOGY

Knowledge Park-II, Greater Noida, UP- 210308

PROGRAM NAME: BTECH(I.T.) SOFTWARE ENGINEERING LAB FILE SUBJECT CODE: BCS651

SESSION: 2024-25

Submitted to:

Dr. Javed Miya

Dept. of Information Technology

Submitted by: Meesam Raza 2200970130066 IT-6B

Experiment-1

Aim: Prepare a SRS document in line with the IEEE recommended standards.

Introduction

1.1 Purpose

This document provides the detailed software requirements for the Online Voting System, which allows users to cast their votes online in a secure and transparent manner.

1.2 Scope

The system will enable registered voters to log in, view candidates, and cast votes. Admin users can manage elections, candidates, and view results. The system aims to minimize fraud and manual errors.

1.3 Definitions, Acronyms, and Abbreviations

-
  - OVS: Online Voting System
  - UI: User Interface
  - OTP: One Time Password

1.4 References

-
  - IEEE 830-1998 SRS format
  - Studocu SE Lab documents

1.5 Overview

This document describes system features, external interfaces, functional and non-functional requirements.

Overall Description

2.1 Product Perspective

OVS is a web-based application, part of the digital election system.

2.2 Product Functions

-
    - Voter registration and login
    - Candidate management
    - Vote casting and result display
    - Secure authentication

2.3 User Classes and Characteristics

-
    - **Admin**: Manages the system and election process.
    - **Voter**: Casts votes.

2.4 Operating Environment

-
    - Web browser (Chrome, Firefox)
    - Server: Apache/MySQL

2.5 Design and Implementation Constraints

-
    - Must ensure secure, one-person-one-vote policy.
    - Should work on standard web platforms.

2.6 User Documentation

-
    - User Manual
    - Help Pages

2.7 Assumptions and Dependencies

-
    - Internet connectivity is required.
    - Voter database is pre-verified.

Specific Requirements

3.1 External Interfaces

-
    - **User Interfaces**: Login page, voting dashboard.
    - **Hardware Interfaces**: None specific (PC/mobile).
    - **Software Interfaces**: Database connection.
    - **Communication Interfaces**: HTTPS protocol.

3.2 Functional Requirements

-
    - FR1: System must allow voter login using credentials.
    - FR2: System must show candidate list.
    - FR3: System must allow casting only one vote per voter.
    - FR4: Admin can add/edit/delete candidates.
    - FR5: System must display election results.

3.3 Non-functional Requirements

-
    - NFR1: The system shall be available 99.9% during election period.
    - NFR2: System response time shall be less than 2 seconds.

- NFR3: System shall enforce secure encryption (SSL/TLS).

System Models

4.1 Use Case Diagram

-
    - Use cases: Login, View Candidates, Cast Vote, View Results, Manage Candidates.

4.2 Data Flow Diagrams (DFD)

-
    - Level 0: Voting process overview
    - Level 1: Voter authentication and vote casting

4.3 UML Diagrams

-
    - Class Diagram
    - Sequence Diagram for vote casting process

Other Requirements

-
    1. **Secure Password Storage**
- Use strong, slow hashing algorithms like bcrypt, Argon2, or PBKDF2.
- Add a unique salt to each password before hashing to prevent rainbow table attacks.
- Store only the hashed passwords in the database — never the plain text.
- Optionally, use a pepper stored separately from the database to add an extra layer of security.
- Implement password strength requirements (length, complexity) to improve user-side security.

OTP Verification for Voting

-
    -
        - Generate a unique, random **OTP (One-Time Password)** when a user attempts to vote.
        - Send the OTP to the user via **email or SMS**, linked to their registered account.
        - Set a **short expiration time** for OTPs (e.g., 5–10 minutes).
        - Allow only **one-time use** for each OTP to prevent reuse.
        - Require OTP entry before finalizing and submitting the vote.
        - Log OTP generation and verification attempts for monitoring fraud or abuse.

Audit Logs for Admin Actions

-
    -
        - **5.3.1** Track all sensitive admin actions, such as:
            - Vote result exports
            - Voter database modifications
            - Candidate management
            - Voting system configuration changes
        - **5.3.2** Each log entry should contain:
            - Admin username or ID
            - Timestamp
            - Action description
            - Affected data (before/after, if applicable)
            - IP address or session details
        - **5.3.3** Store logs in a **tamper-evident** or immutable format.
        - **5.3.4** Regularly review logs for suspicious activity or errors.
        - **5.3.5** Provide role-based access to logs (e.g., view-only for auditors, full access for super admins).

Use Case Diagram for Online Voting System

Actors:

- **Admin**
- **Voter Use Cases:**
- Login

- Register
- View Candidates
- Cast Vote
- View Results
- Manage Candidates

Roles of Actors

Admin:

- Manages candidate details.
- Starts and ends voting sessions.
- Views and declares election results.

Voter:

- Registers and logs into the system.
- Views the list of candidates.
- Casts a vote.
- Views results after voting session ends.

Use Case Specifications

Use Case: Login

-
    - **Actors:** Voter, Admin
    - **Precondition:** The user must be registered.
    - **Postcondition:** The user is authenticated and redirected to their dashboard.
    - **Function:** Verifies user credentials and grants access.

Use Case: Register

-
    - **Actors:** Voter
    - **Precondition:** User is not yet registered.
    - **Postcondition:** User information is stored and account is created.
    - **Function:** Allows voters to create an account in the system.

Use Case: View Candidates

-
    - **Actors:** Voter
    - **Precondition:** Voter is logged in.
    - **Postcondition:** List of current candidates is displayed.
    - **Function:** Displays all candidates standing for election.

Use Case: Cast Vote

-
    - **Actors:** Voter
    - **Precondition:** Voter is authenticated and voting session is active.
    - **Postcondition:** Vote is recorded and voter cannot vote again.
    - **Function:** Allows voter to select a candidate and submit a vote.

Use Case: View Results

-
    - **Actors:** Voter, Admin
    - **Precondition:** Voting session has ended.
    - **Postcondition:** Election results are displayed.
    - **Function:** Shows the result of the election to the users.

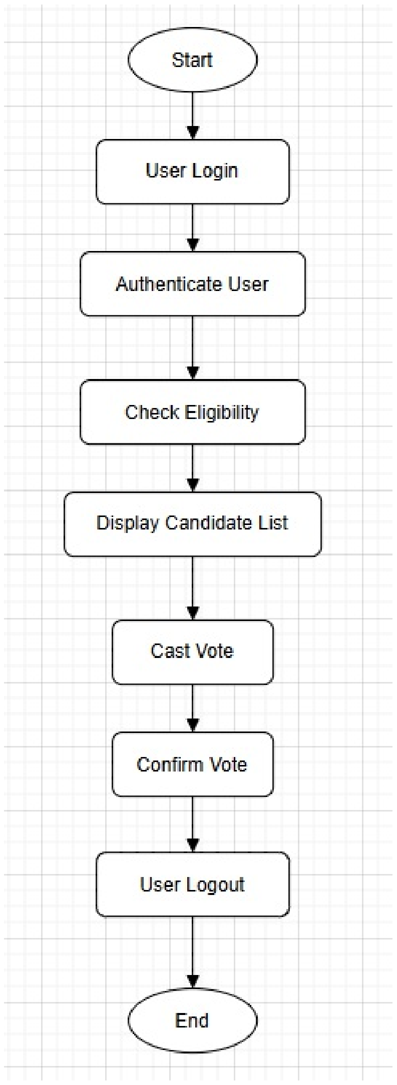
Use Case: Manage Candidates

-
    - **Actors:** Admin

- **Precondition:** Admin is authenticated.
- **Postcondition:** Candidates list is updated.
- **Function:** Allows admin to add, edit, or delete candidate information.

Experiment-3

Aim: Draw the activity diagram.



Components of an Activity Diagram

An Activity Diagram is a type of UML behavioral diagram used to represent the dynamic aspects of a system. It illustrates the flow of control or data from one activity to another within a process.

Initial Node

-
  - Symbol: Solid filled circle.
  - Description: Represents the starting point of the activity flow. Every activity diagram must have exactly one initial node.

Activity (or Action)

-
  - Symbol: Rounded rectangle.
  - Description: Denotes a specific task or operation performed in the process. It is the fundamental unit of work in an activity diagram.

Decision Node

-
  - Symbol: Diamond.
  - Description: Represents a point where the flow can branch into multiple paths based on conditions or decisions.

Merge Node

-
  - Symbol: Diamond.

    - Description: Combines multiple alternative flows into a single outgoing flow. It is used to reunify decision branches.

Fork Node

-
    - Symbol: Thick horizontal or vertical bar.
    - Description: Splits a single flow into two or more parallel flows, enabling concurrent execution of activities.

Join Node

-
    - Symbol: Thick horizontal or vertical bar.
    - Description: Synchronizes two or more concurrent flows into a single subsequent flow.

Control Flow

-
    - Symbol: Solid arrow.
    - Description: Indicates the order in which activities are performed. It connects actions, decisions, and other elements in the diagram.

Final Node

-
    - Symbol: Bullseye (a solid circle inside a hollow circle).
    - Description: Marks the end of the activity flow. An activity diagram can have more than one final node, depending on the process design.

Experiment-4

Aim: Identify the classes. Classify them as weak and strong classes and draw the class diagram.

For an online voting system, the system might consist of several classes that interact with each other. Here's a breakdown of the classes typically involved, and we can classify them as weak or strong based on their cohesion, responsibilities, and interactions.

User (Strong Class)

- **Responsibilities**: Represents a voter or an admin in the system. Handles user authentication, registration, and profile management.
- **Attributes**: user_id, name, email, password, role
- **Methods**: login(), register(), updateProfile(), logout()
- **Reason**: The User class is self-contained, handling all user-related functionalities and representing a clear concept.

Voting Session (Strong Class)

- **Responsibilities**: Manages a voting session, including vote casting, closing of polls, and result generation.
- **Attributes**: session_id, start_time, end_time, is_open
- **Methods**: startSession(), endSession(), castVote(), getResults()
- **Reason**: The VotingSession class encapsulates the logic of running a voting session, handling one specific responsibility, so it's considered strong.

Candidate (Strong Class)

- **Responsibilities**: Represents a candidate in the election, including information like their name, description, and the votes they received.
- **Attributes**: candidate_id, name, party, votes_received
- **Methods**: updateVotes()
- **Reason**: The Candidate class is highly cohesive as it encapsulates everything related to a candidate in the election.

Vote (Weak Class)

- **Responsibilities**: Represents an individual vote cast by a user in a particular voting session.
- **Attributes**: vote_id, user_id, candidate_id, session_id
- **Methods**: validateVote()
- **Reason**: While this class has its purpose, it could be seen as weak because it is highly dependent on other classes (User, Candidate, VotingSession) and might not have enough responsibility on its own.

Election (Strong Class)

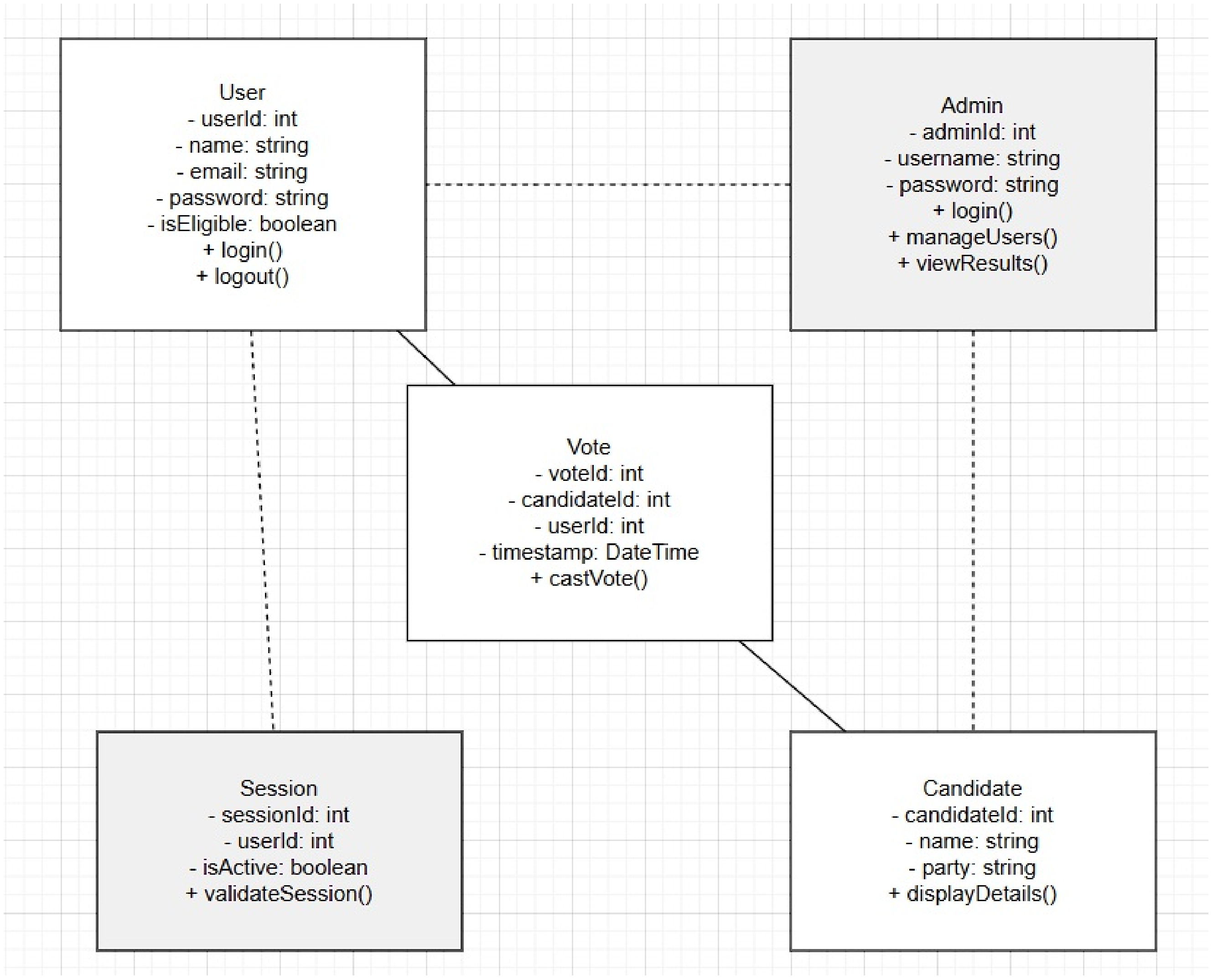
- **Responsibilities**: Represents the overall election process, including managing different voting sessions, candidates, and generating final results.
- **Attributes**: election_id, start_date, end_date, list_of_sessions
- **Methods**: createVotingSession(), closeElection(), generateFinalResults()
- **Reason**: The Election class encapsulates the entire election process, making it strong.

Admin (Weak Class)

- **Responsibilities**: Manages the administrative actions in the system, such as approving candidates, managing users, and viewing election results.
- **Attributes**: admin_id, role, permissions
- **Methods**: approveCandidate(), viewResults(), blockUser()
- **Reason**: The Admin class may be weak depending on how it's structured, as it could overlap with other functionalities such as the User class and has external dependencies.

AuditLog (Weak Class)

- **Responsibilities**: Records actions performed by users and admins within the system (like voting, registering, and session changes).
- **Attributes**: log_id, timestamp, action, user_id
- **Methods**: recordAction(), getLogs()
- **Reason**: The AuditLog class is more of a supporting class and could be classified as weak since it has minimal responsibility and is highly dependent on other classes for functionality.



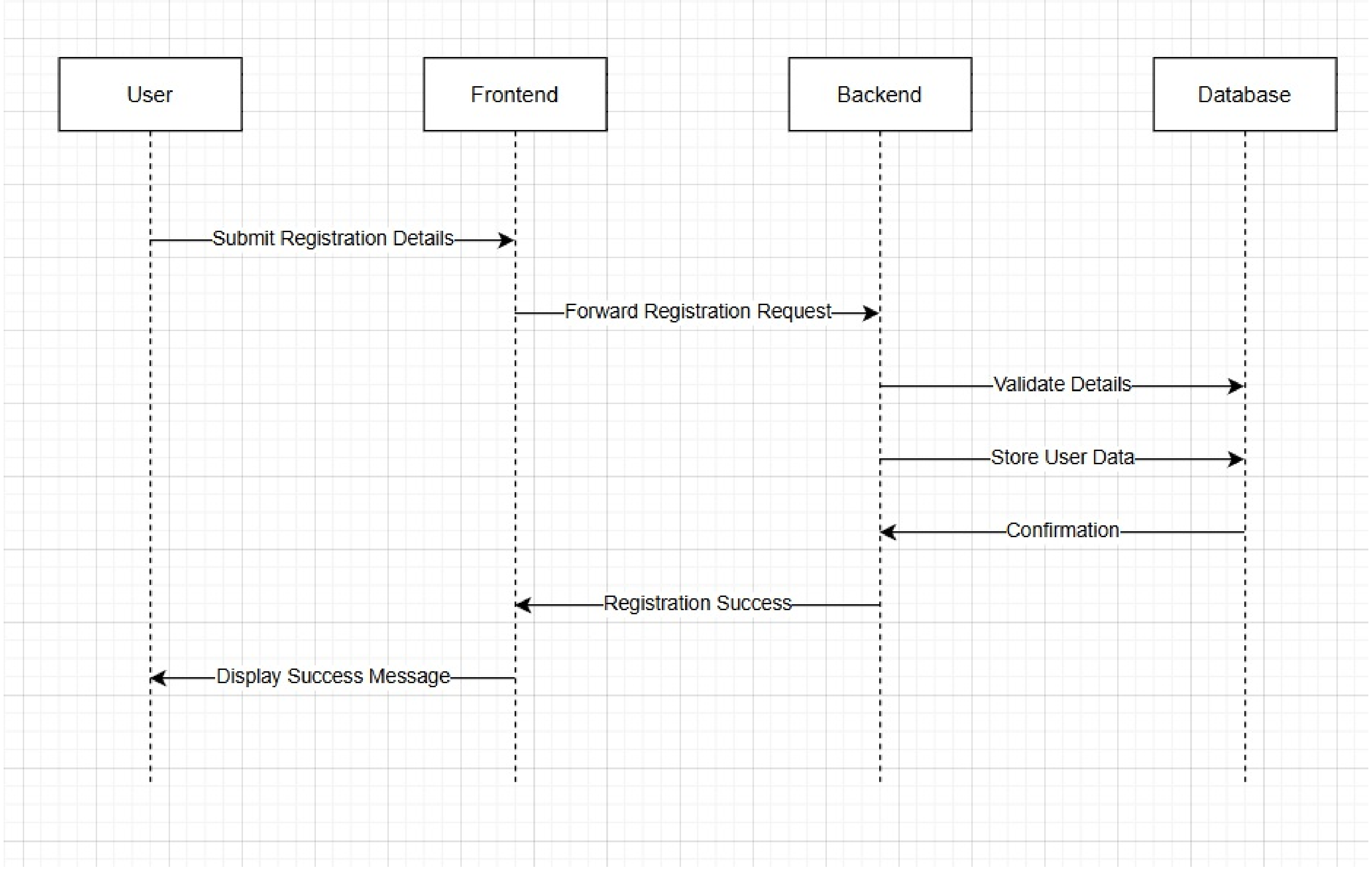
Experiment-5

Aim: Draw the sequence diagram for any two scenarios.

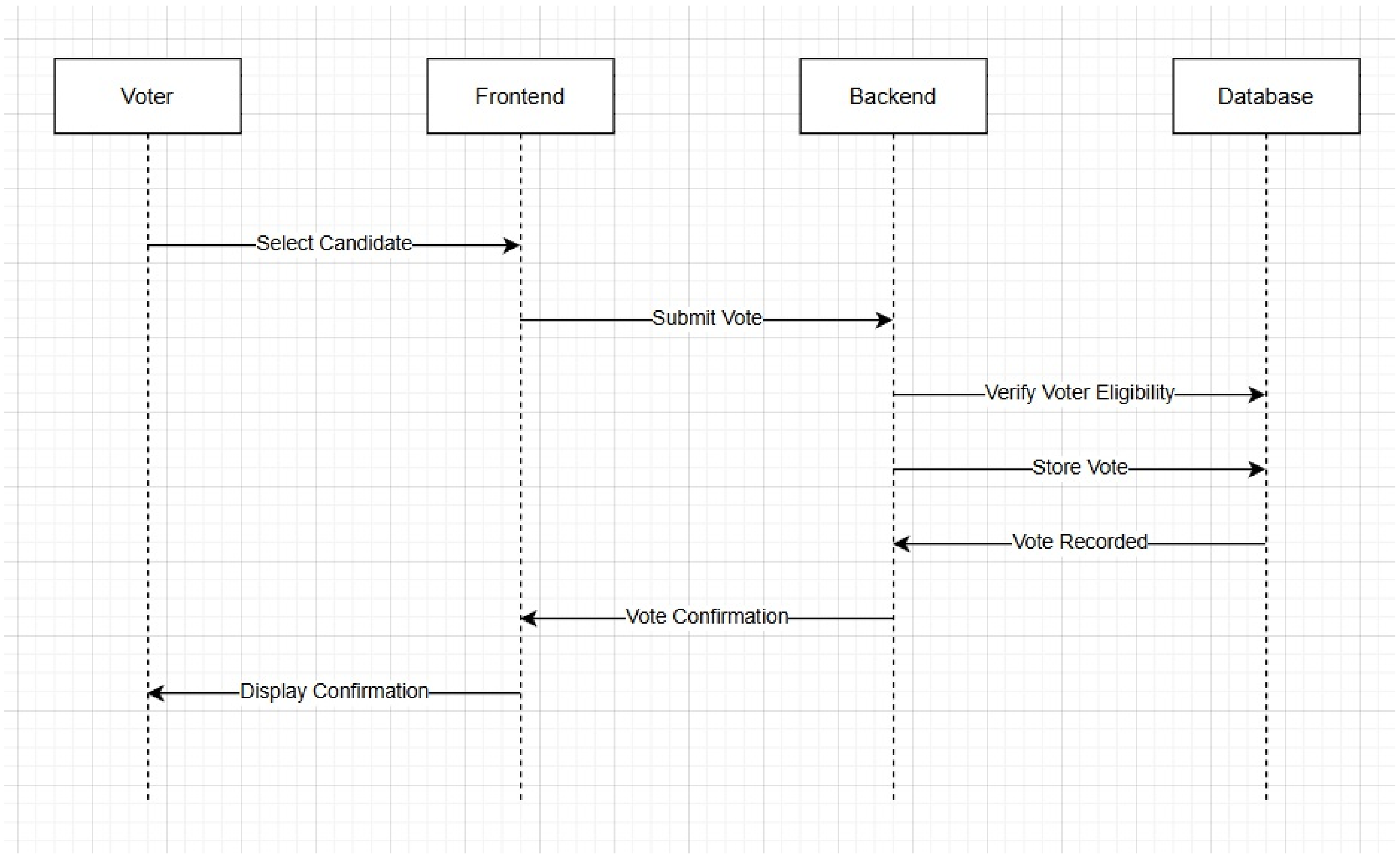
A sequence diagram is a type of interaction diagram in UML (Unified Modelling Language) that shows how objects interact in a particular scenario of a use case or system operation. It represents the sequence of messages exchanged between objects or components within a system over time.

Sequence diagrams are particularly useful for visualizing the dynamic behavior of a system or a specific use case. They help in understanding the flow of control and the sequence of interactions between different components or objects involved in a scenario. Sequence diagrams are often used during system design and development phases to clarify and specify the interaction details before implementation.

Scenario 1: Voter Registration



Scenario 2: Casting a Vote



Registered voter casting their vote

Experiment-6

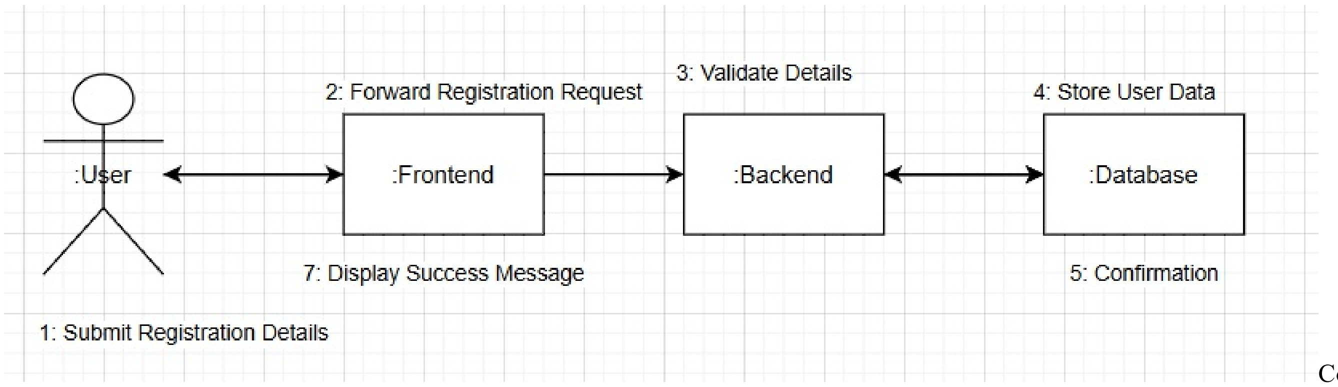
Aim: Draw the collaboration diagram

Collaboration Diagram:

A Collaboration Diagram (also known as a Communication Diagram) is a type of UML interaction diagram that focuses on the structural organization of objects that send and receive messages. It shows how objects interact with each other to perform a behavior or operation, emphasizing the relationships among objects.

It combines features of both class diagrams and sequence diagrams, highlighting both the interaction sequence and the object links involved.

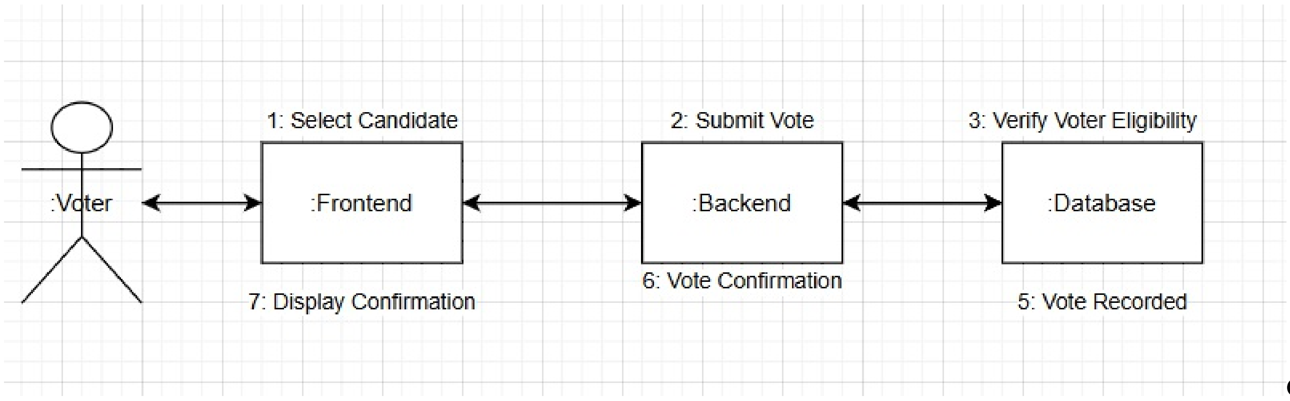
Scenario 1: Voter Registration Process



3: Validate Details
2: Forward Registration Request
4: Store User Data
:User
:Frontend
:Backend
:Database
7: Display Success Message
5: Confirmation
1: Submit Registration Details

Collaboration Diagram for User Voting Process:

Scenario 2: Casting a Vote



Collaboration Diagram for casting the vote

Experiment-7

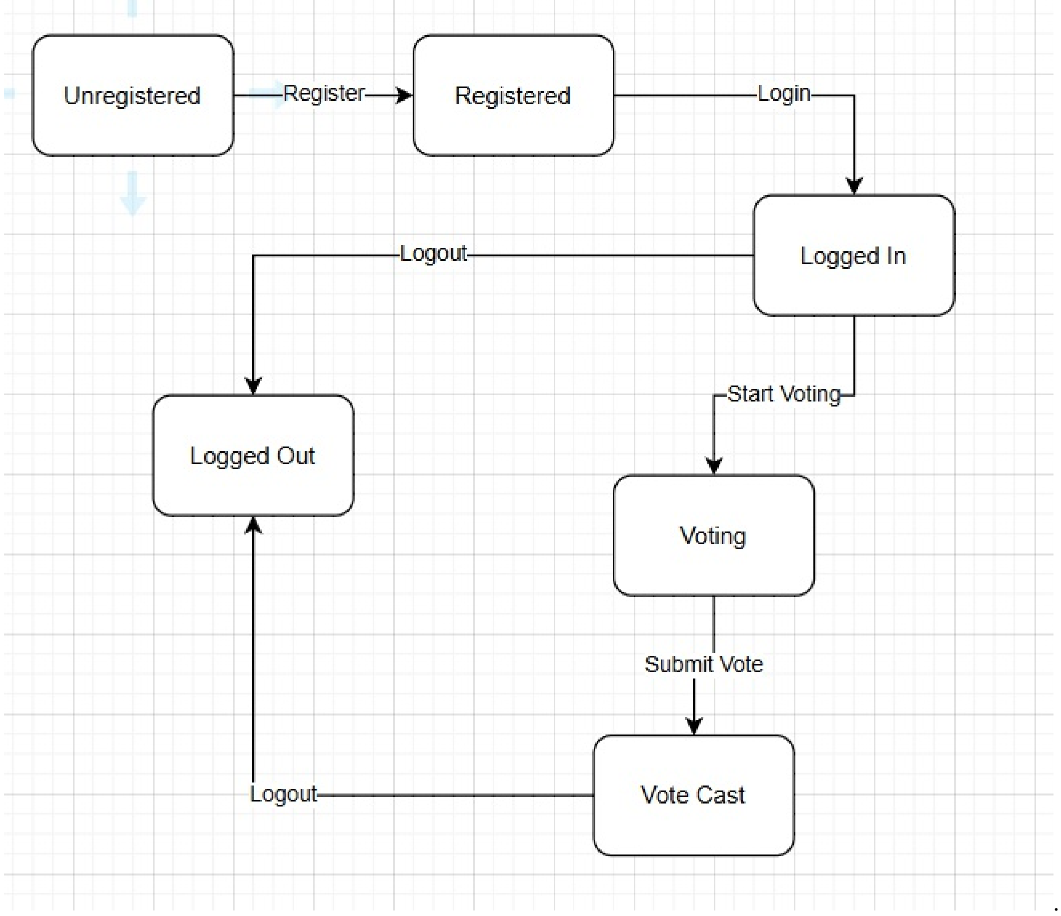
Aim: Draw the state chart diagram

A state chart diagram (or state machine diagram) shows the various states that an object goes through during its lifecycle, as well as the events that cause it to transition from one state to another. In an online voting system, various entities (such as a User or VotingSession) could have state charts to represent the different phases in their interaction.

Below are state chart diagrams for two different entities in an online voting system:

Scenario 1: State Chart for User

This diagram represents the different states a User can be in, from registration to casting a vote and logging out.



Explanation:

1. Not Registered: The User starts in the Not Registered state. The user can register to move to the Registered state.
2. Registered: After registration, the User is in the Registered state. They can log in, which transitions them to the Logged In state.
3. Logged In: Once logged in, the User can participate in the voting process. They can select a candidate and

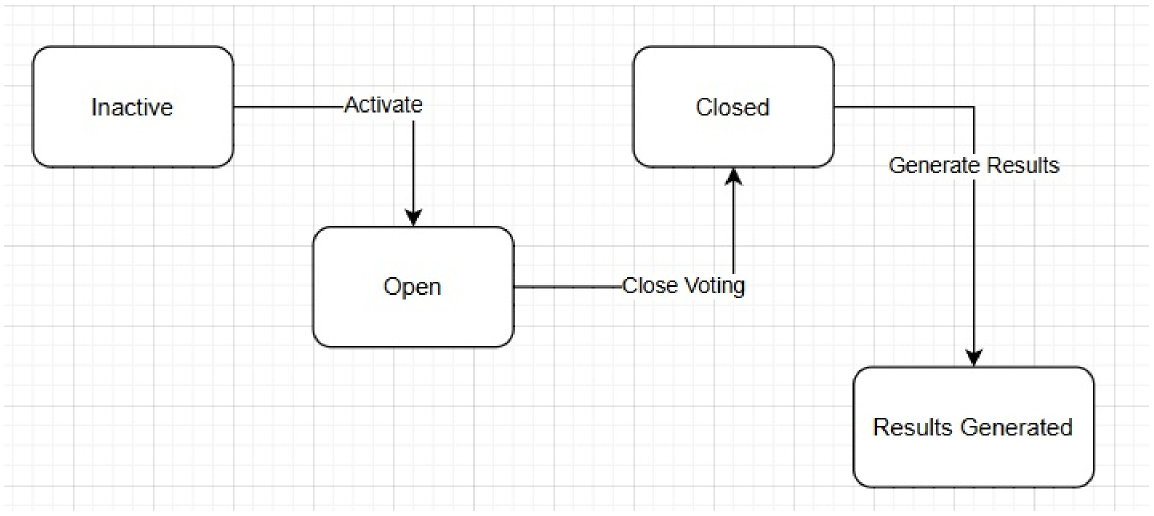
cast their vote, which takes them to the Casting Vote state.

1. Casting Vote: After the user casts their vote, they return to the Logged In state or can log out, which takes them back to the Logged Out state.
2. Logged Out: The User logs out, transitioning back to the Not Registered state (if they were not previously registered) or just remain logged out.

Scenario 2: State Chart for Voting Session

This diagram represents the states of a VotingSession in an online voting system, such as when it is open, closed, or results are being generated.

State Chart Diagram for Voting Session:



Explanation:

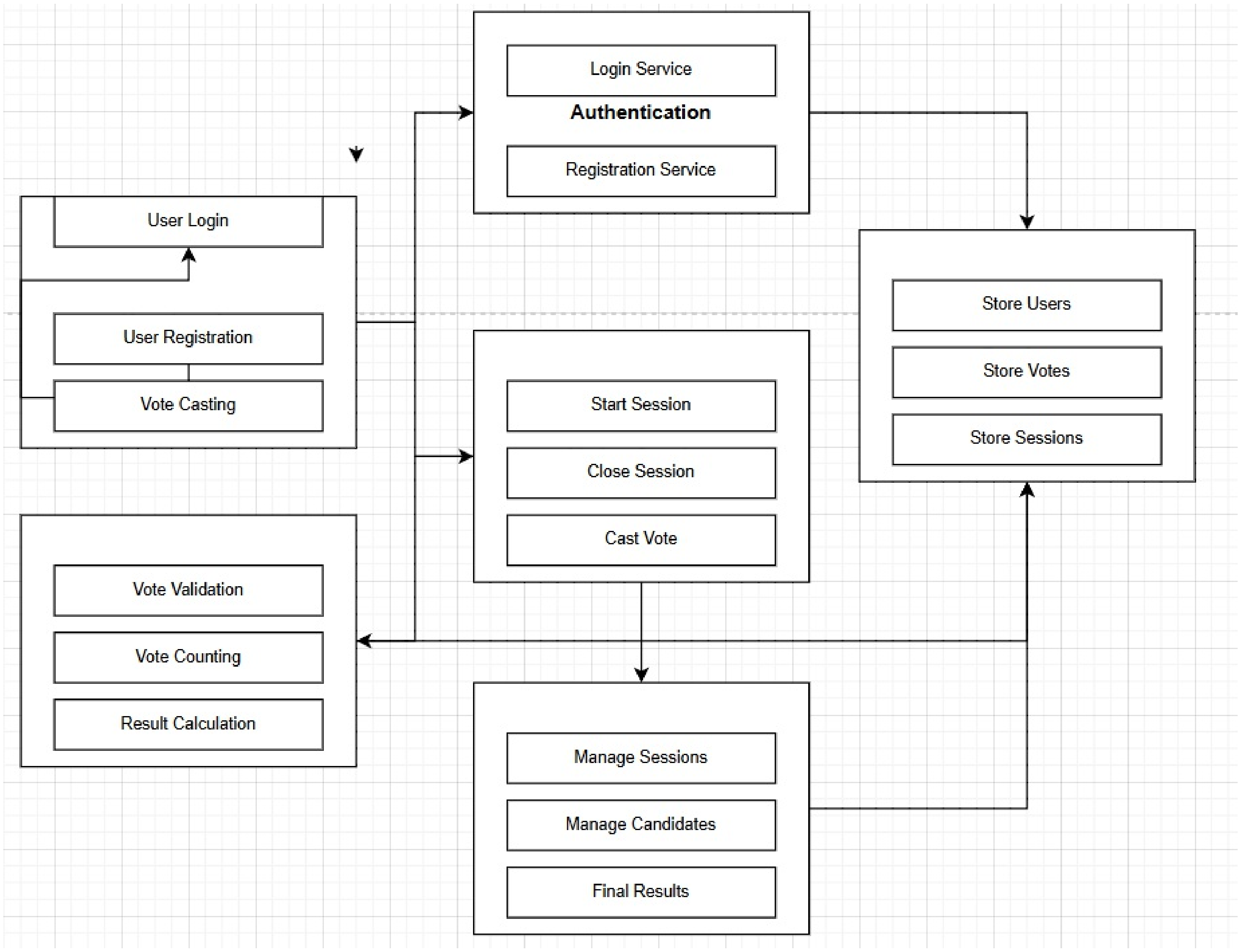
1. Not Started: The VotingSession starts in the Not Started state. It is in this state before the session begins, and it transitions to the Opened state once the session starts.
2. Opened: The VotingSession is open for users to cast their votes. The session remains in this state while voting is ongoing.
3. Closed: Once voting ends, the session moves to the Closed state, indicating no further votes can be cast.
4. Generating Results: After the session is closed, the system generates the results based on the votes cast.

5. Not Started: Once the session is completed, the system may return to the Not Started state to prepare for a new voting session

Experiment-8

Aim: Draw the component diagram.

A component diagram is used to visualize the components or modules of a system and their relationships or dependencies. It focuses on the system's architecture and how various software components interact with one another.



Explanation of Components:

User Interface Component:

-
  1. **Responsibilities**: Provides the front-end interface for users to interact with the voting system.

Subcomponents:

-
  -
    1. **User Login**: Handles user login functionality.
    2. **User Registration**: Allows new users to register.
    3. **Vote Casting**: Provides the interface for users to cast their votes.

Authentication Component:

-
  1. **Responsibilities**: Handles user authentication, such as login and registration.

Subcomponents:

-
  -
    1. **Login Service**: Verifies user credentials.

2. **Registration Service**: Handles user sign-up and registration.

Voting Session Component:

-
    1. **Responsibilities**: Manages the voting session, including starting, closing, and handling the vote casting process.

Subcomponents:

-
    -
        1. **Start Session**: Begins the voting session.
        2. **Close Session**: Ends the voting session.
        3. **Cast Vote**: Manages the process of a user casting a vote.

Voting Logic Component:

-
    1. **Responsibilities**: Contains the core logic for vote validation, counting, and result generation.

Subcomponents:

-
    -
        1. **Vote Validation**: Verifies that the vote is legitimate.
        2. **Vote Counting**: Counts the number of votes for each candidate.
        3. **Result Calculation**: Calculates the results based on the votes.

Election Component:

-
    1. **Responsibilities**: Manages the overall election process, including managing multiple voting sessions and generating final results.

Subcomponents:

-
    -
        1. **Manage Sessions**: Handles the creation and management of voting sessions.
        2. **Manage Candidates**: Manages the candidates participating in the election.
        3. **Final Results**: Handles the aggregation and presentation of final results.

Database Component:

-
    1. **Responsibilities**: Stores all the data related to users, votes, sessions, and election information.

Subcomponents:

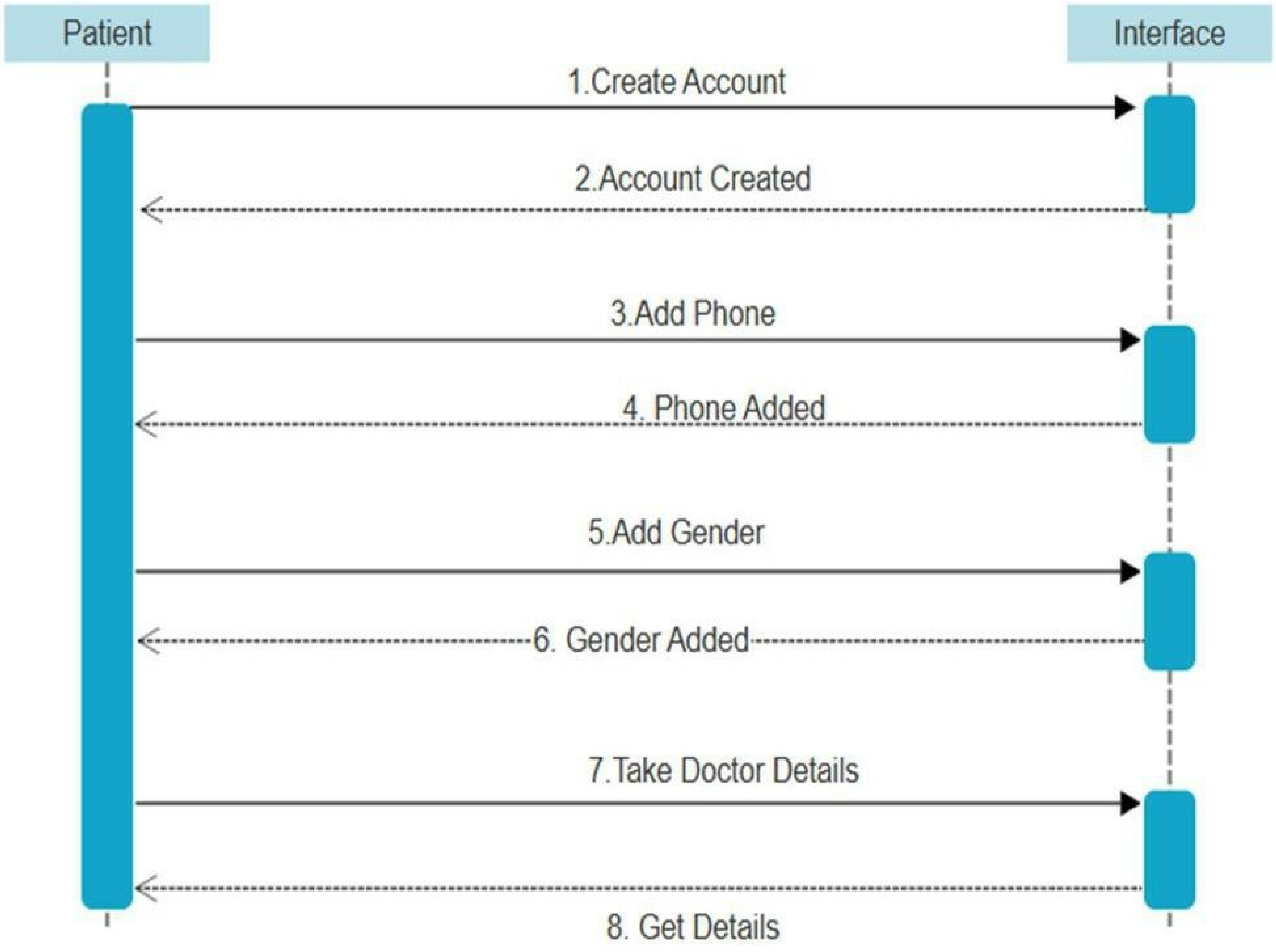
-
    -
        1. **Store Users**: Stores user data.
        2. **Store Votes**: Stores the votes cast during elections.
        3. **Store Sessions**: Stores details about the voting sessions.

Experiment-5

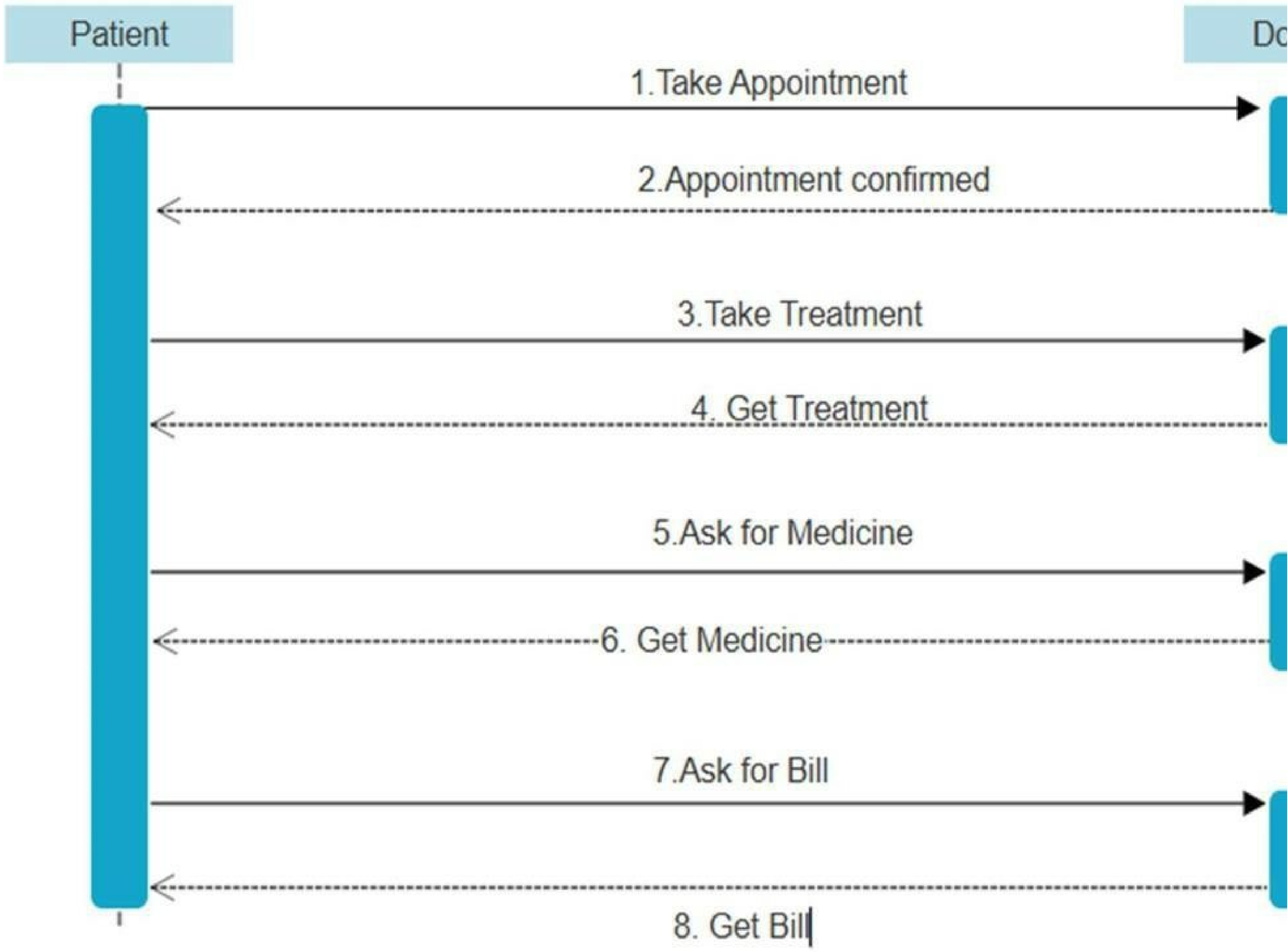
Aim: Draw the sequence diagram for any two scenarios.

A sequence diagram is a type of interaction diagram in UML (Unified Modelling Language) that shows how objects interact in a particular scenario of a use case or system operation. It represents the sequence of messages exchanged between objects or components within a system over time.

Sequence diagrams are particularly useful for visualizing the dynamic behavior of a system or a specific use case. They help in understanding the flow of control and the sequence of interactions between different components or objects involved in a scenario. Sequence diagrams are often used during system design and development phases to clarify and specify the interaction details before implementation



Sequence Diagram for Patient interface for Doctor Details (Scenario-1)



Sequence Diagram for Patient and Doctor for Treatment (Scenario-2)

Components of a Collaboration Diagram

1. **Objects**
    - Represent instances of classes.
    - Denoted as: objectName:ClassName
    - Shown within rectangles.

Links (Associations)

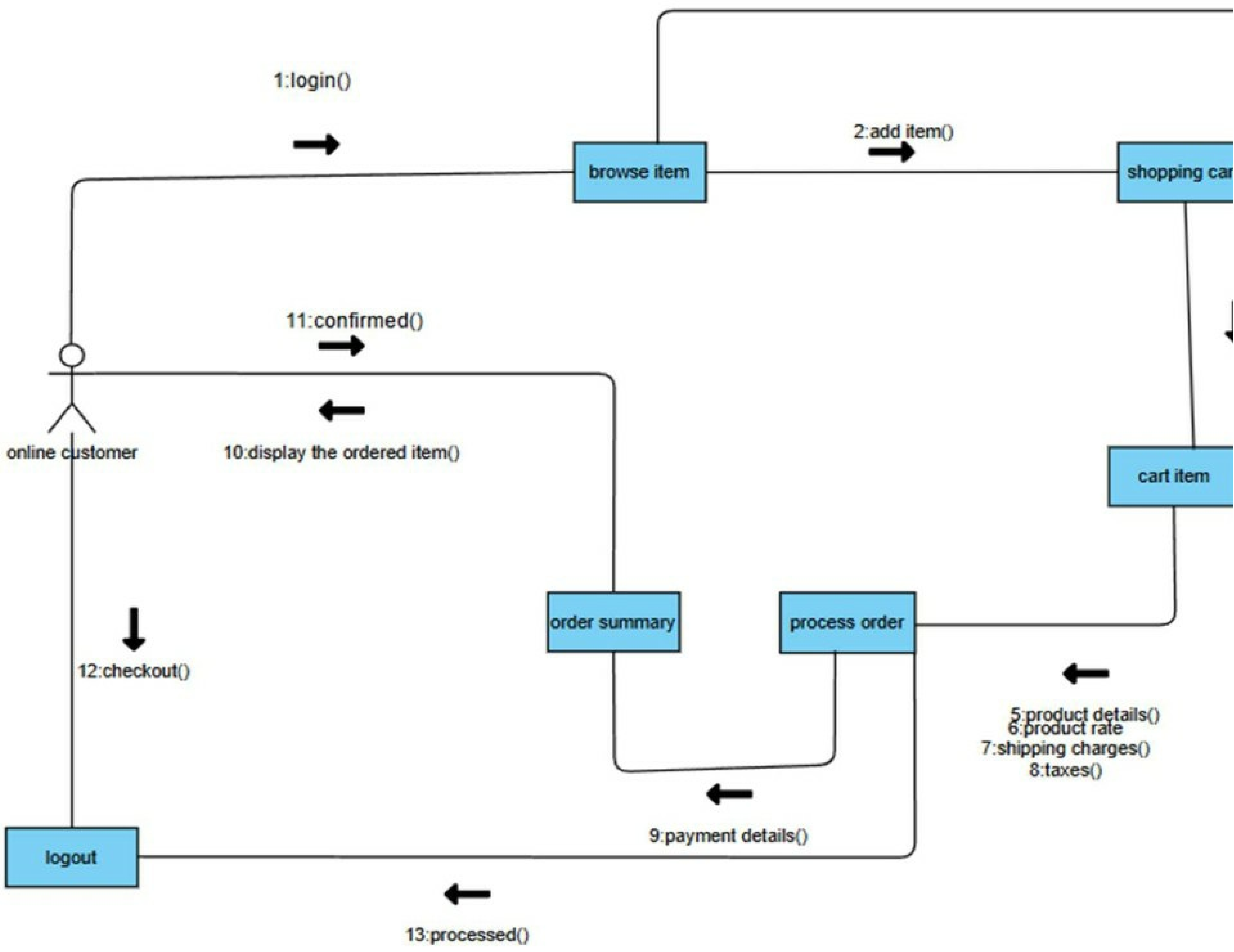
-
    - Solid lines connecting objects to indicate relationships or communication paths.
    - Represent that objects can send messages to each other.

Messages

-
    - Indicate interaction between objects.
    - Labeled arrows placed along links.
    - Messages are numbered sequentially to represent the order of execution.
    - Syntax: sequenceNumber: messageName(parameters)

Multiplicity (Optional)

-
    - Indicates how many instances of an object are involved in the interaction.


1:login()
browse item
2:add item()
shopping car
11:confirmed()
online customer
10:display the ordered item()
cart item
12:checkout()
order summary
process order
5:product details()
6:product rate
7:shipping charges()
8:taxes()
9:payment details()
logout
13:processed()