TUTORIAL NO: 06 UML Diagrams: Use Case Diagram

THEORY:

What is a use case diagram?

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system
- When to apply use case diagrams

A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case.

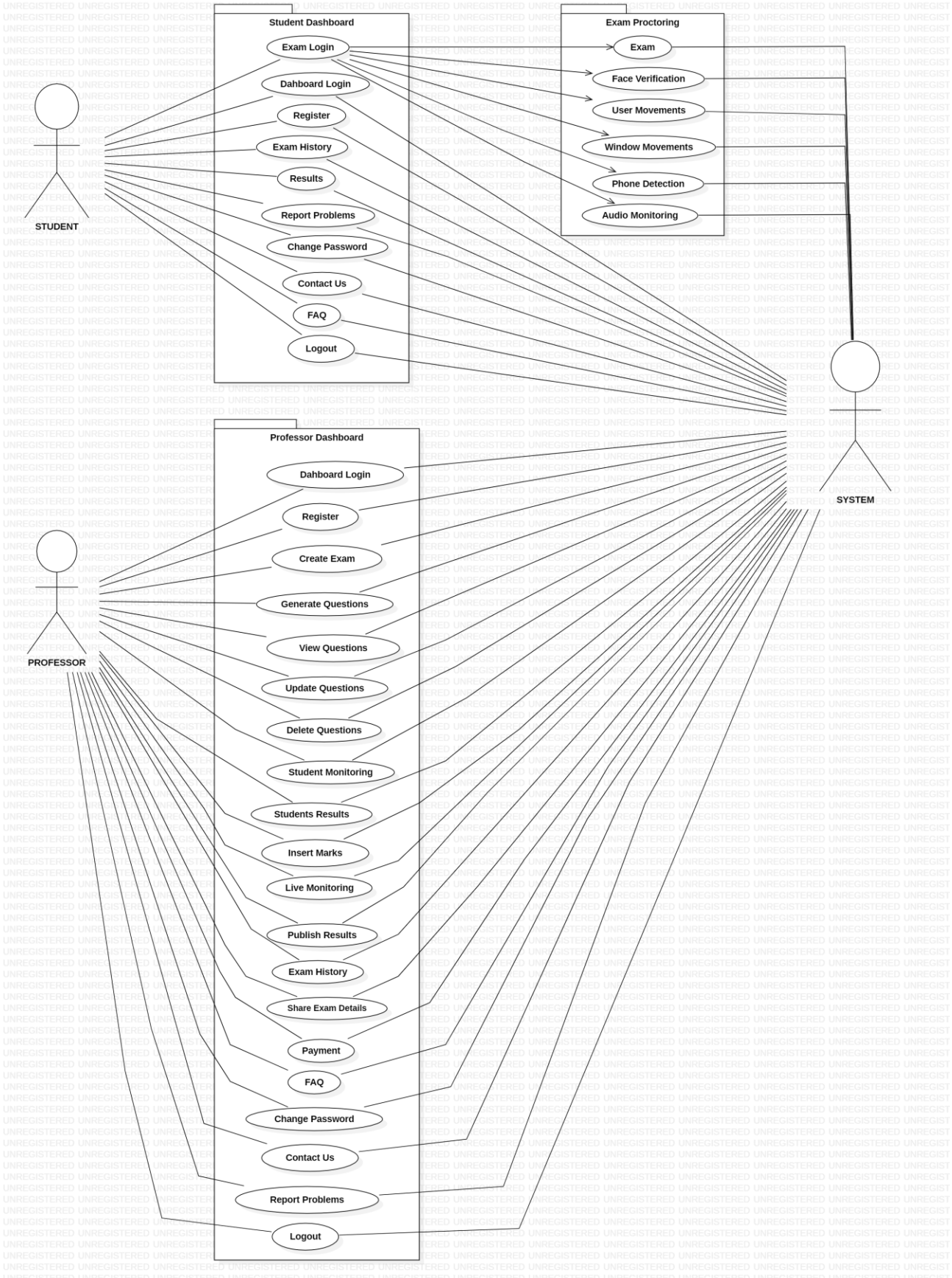
UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case
- Use case diagram components

Common components include:

- Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- System: A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- Goals: The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.


Student Dashboard
Exam Login
Dahboard Login
Register
Exam History
Results
Report Problems
Change Password
Contact Us
FAQ
Logout
Exam Proctoring
Exam
Face Verification
User Movements
Window Movements
Phone Detection
Audio Monitoring
STUDENT
SYSTEM
Professor Dashboard
Dahboard Login
Register
Create Exam
Generate Questions
View Questions
Update Questions
Delete Questions
Student Monitoring
Students Results
Insert Marks
Live Monitoring
Publish Results
Exam History
Share Exam Details
Payment
FAQ
Change Password
Contact Us
Report Problems
Logout
PROFESSOR

TUTORIAL NO: 07 UML Diagrams: Class Diagram

THEORY:

What is a class diagram in UML?

The Unified Modeling Language (UML) can help you model systems in various ways. One of the more popular types in UML is the class diagram. Popular among software engineers to document software architecture, class diagrams are a type of structure diagram because they describe what must be present in the system being modeled.

UML was set up as a standardized model to describe an object-oriented programming approach. Since classes are the building block of objects, class diagrams are the building blocks of UML. The various components in a class diagram can represent the classes that will actually be programmed, the main objects, or the interactions between classes and objects.

The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section expresses the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

Benefits of class diagrams

Class diagrams offer a number of benefits for any organization. Use UML class diagrams to:

- Illustrate data models for information systems, no matter how simple or complex.
- Better understand the general overview of the schematics of an application.
- Visually express any specific needs of a system and disseminate that information throughout the business.
- Create detailed charts that highlight any specific code needed to be programmed and implemented to the described structure.
- Provide an implementation-independent description of types used in a system that are later passed between its components.

Basic components of a class diagram

The standard class diagram is composed of three sections:

- **Upper section:** Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- **Middle section:** Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.

- **Bottom section:** Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data.

Member access modifiers
All classes have different access levels depending on the access modifier (visibility). Here are the access levels with their corresponding symbols:

- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

Member scopes
There are two scopes for members: classifiers and instances.

Classifiers are static members while instances are the specific instances of the class. If you are familiar with basic OO theory, this isn't anything groundbreaking.

Additional class diagram components
Depending on the context, classes in a class diagram can represent the main objects, interactions in the application, or classes to be programmed. To answer the question "What is a class diagram in UML?" you should first understand its basic makeup.

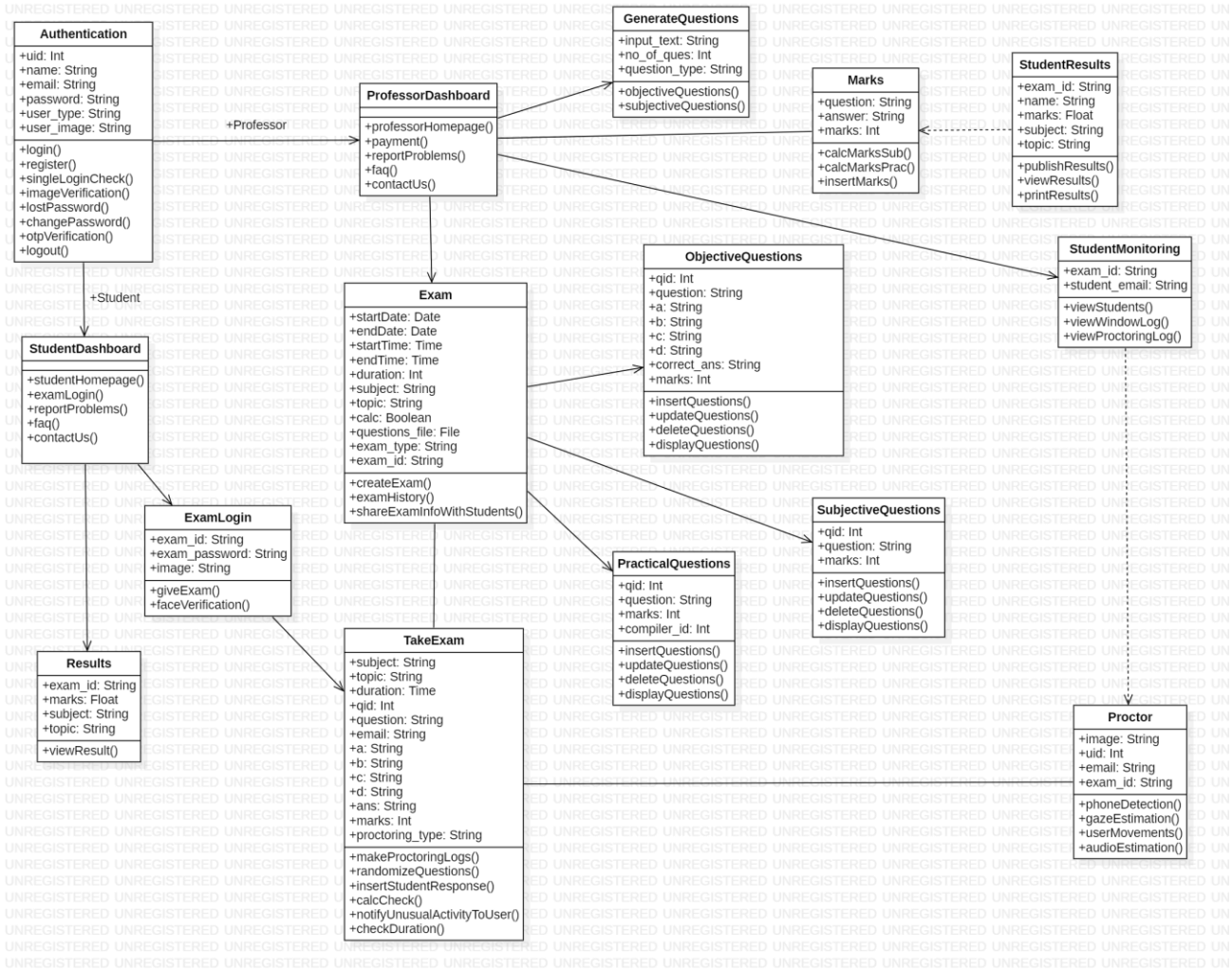
- **Classes:** A template for creating objects and implementing behavior in a system. In UML, a class represents an object or a set of objects that share a common structure and behavior. They're represented by a rectangle that includes rows of the class name, its attributes, and its operations. When you draw a class in a class diagram, you're only required to fill out the top row—the others are optional if you'd like to provide more detail.
    - **Name:** The first row in a class shape.
    - **Attributes:** The second row in a class shape. Each attribute of the class is displayed on a separate line.
    - **Methods:** The third row in a class shape. Also known as operations, methods are displayed in list format with each operation on its own line.

- **Signals**: Symbols that represent one-way, asynchronous communications between active objects.
- **Data types:** Classifiers that define data values. Data types can model both primitive types and enumerations.
- **Packages:** Shapes designed to organize related classifiers in a diagram. They are symbolized with a large tabbed rectangle shape.
- **Interfaces:** A collection of operation signatures and/or attribute definitions that define a cohesive set of behaviors. Interfaces are similar to classes, except that a class can have an instance of its type, and an interface must have at least one class to implement it.
- **Enumerations:** Representations of user-defined data types. An enumeration includes groups of identifiers that represent values of the enumeration.
- **Objects:** Instances of a class or classes. Objects can be added to a class diagram to represent either concrete or prototypical instances.
- **Artifacts:** Model elements that represent the concrete entities in a software system, such as documents, databases, executable files, software components, etc.

Interactions

The term "interactions" refers to the various relationships and links that can exist in class and object diagrams. Some of the most common interactions include:

- **Inheritance:** The process of a child or sub-class taking on the functionality of a parent or superclass, also known as generalization. It's symbolized with a straight connected line with a closed arrowhead pointing towards the superclass.


Authentication
+uid: Int
+name: String
+email: String
+password: String
+user_type: String
+user_image: String
+login()
+register()
+singleLoginCheck()
+imageVerification()
+lostPassword()
+changePassword()
+otpVerification()
+logout()
+Professor
+Student
ProfessorDashboard
+professorHomepage()
+payment()
+reportProblems()
+faq()
+contactUs()
GenerateQuestions
+input_text: String
+no_of_ques: Int
+question_type: String
+objectiveQuestions()
+subjectiveQuestions()
Marks
+question: String
+answer: String
+marks: Int
+calcMarksSub()
+calcMarksPrac()
+insertMarks()
StudentResults
+exam_id: String
+name: String
+marks: Float
+subject: String
+topic: String
+publishResults()
+viewResults()
+printResults()
StudentMonitoring
+exam_id: String
+student_email: String
+viewStudents()
+viewWindowLog()
+viewProctoringLog()
StudentDashboard
+studentHomepage()
+examLogin()
+reportProblems()
+faq()
+contactUs()
Exam
+startDate: Date
+endDate: Date
+startTime: Time
+endTime: Time
+duration: Int
+subject: String
+topic: String
+calc: Boolean
+questions_file: File
+exam_type: String
+exam_id: String
+createExam()
+examHistory()
+shareExamInfoWithStudents()
ObjectiveQuestions
+qid: Int
+question: String
+a: String
+b: String
+c: String
+d: String
+correct_ans: String
+marks: Int
+insertQuestions()
+updateQuestions()
+deleteQuestions()
+displayQuestions()
SubjectiveQuestions
+qid: Int
+question: String
+marks: Int
+insertQuestions()
+updateQuestions()
+deleteQuestions()
+displayQuestions()
PracticalQuestions
+qid: Int
+question: String
+marks: Int
+compiler_id: Int
+insertQuestions()
+updateQuestions()
+deleteQuestions()
+displayQuestions()
ExamLogin
+exam_id: String
+exam_password: String
+image: String
+giveExam()
+faceVerification()
TakeExam
+subject: String
+topic: String
+duration: Time
+qid: Int
+question: String
+email: String
+a: String
+b: String
+c: String
+d: String
+ans: String
+marks: Int
+proctoring_type: String
+makeProctoringLogs()
+randomizeQuestions()
+insertStudentResponse()
+calcCheck()
+notifyUnusualActivityToUser()
+checkDuration()
Results
+exam_id: String
+marks: Float
+subject: String
+topic: String
+viewResult()
Proctor
+image: String
+uid: Int
+email: String
+exam_id: String
+phoneDetection()
+gazeEstimation()
+userMovements()
+audioEstimation()

TUTORIAL NO: 08 UML Diagrams: Activity Diagram

What is an activity diagram?

The Unified Modeling Language includes several subsets of diagrams, including structure diagrams, interaction diagrams, and behavior diagrams. Activity diagrams, along with use case and state machine diagrams, are considered behavior diagrams because they describe what must happen in the system being modeled.

Stakeholders have many issues to manage, so it's important to communicate with clarity and brevity. Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior. You'll use a set of specialized symbols—including those used for starting, ending, merging, or receiving steps in the flow—to make an activity diagram, which we'll cover in more depth within this activity diagram guide.

Benefits of activity diagrams

Activity diagrams present a number of benefits to users. Consider creating an activity diagram to:

- Demonstrate the logic of an algorithm.
- Describe the steps performed in a UML use case.
- Illustrate a business process or workflow between users and the system.
- Simplify and improve any process by clarifying complicated use cases.
- Model software architecture elements, such as method, function, and operation.

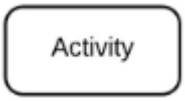
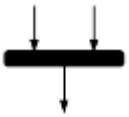
Basic components of an activity diagram

Before you begin making an activity diagram, you should first understand its makeup. Some of the most common components of an activity diagram include:

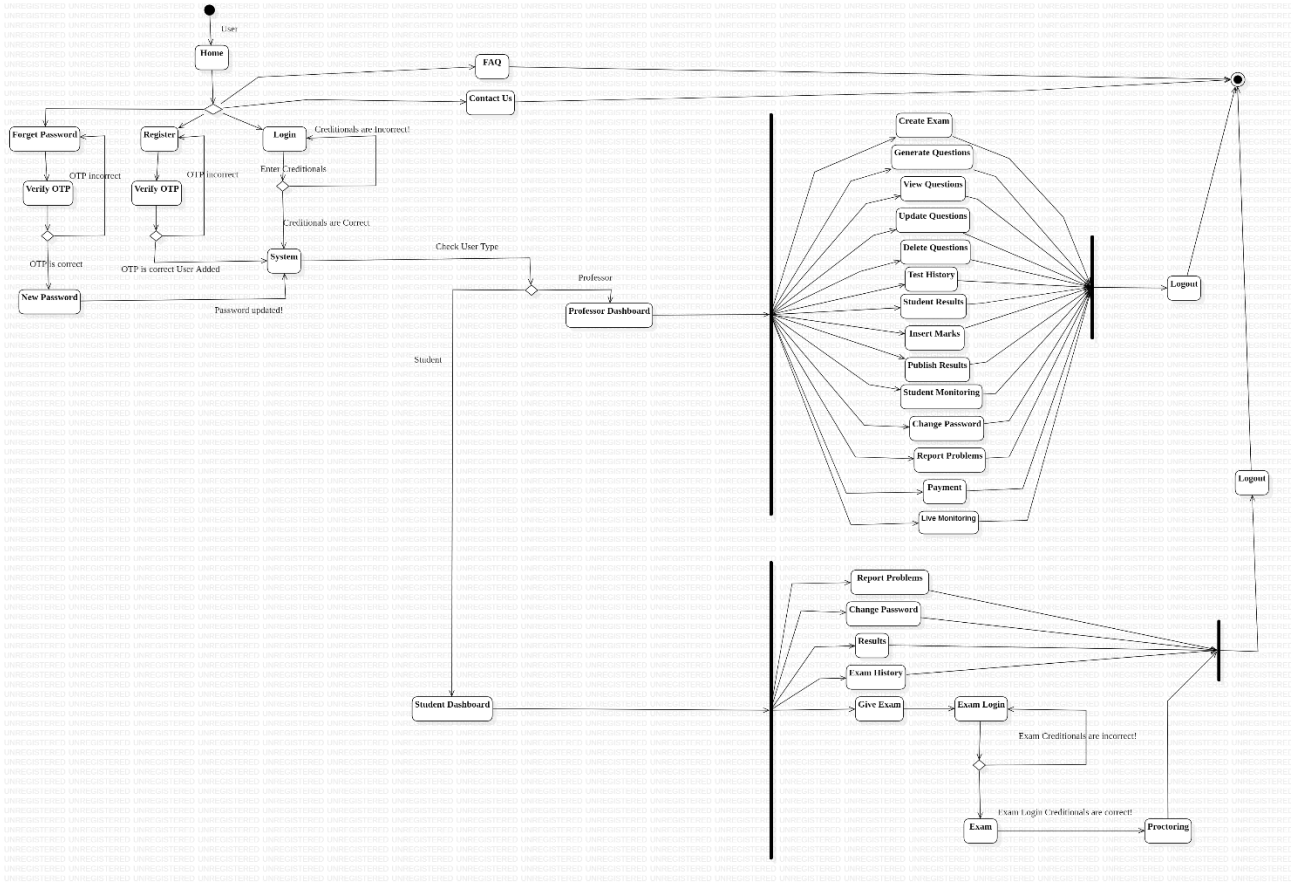
- **Action:** A step in the activity wherein the users or software perform a given task. Actions are symbolized with round-edged rectangles.
- **Decision node:** A conditional branch in the flow that is represented by a diamond. It includes a single input and two or more outputs.
- **Control flows:** Another name for the connectors that show the flow between steps in the diagram.
- **Start node:** Symbolizes the beginning of the activity. The start node is represented by a black circle.
- **End node:** Represents the final step in the activity. The end node is represented by an outlined black circle.

Activity diagram symbols

These activity diagram shapes and symbols are some of the most common types you'll find in UML diagrams.

Symbol	Name	Description
	Start symbol	Represents the beginning of a process or workflow in an activity diagram. It can be used by itself or with a note symbol that explains the starting point.
	Activity symbol	Indicates the activities that make up a modeled process. These symbols, which include short descriptions within the shape, are the main building blocks of an activity diagram.
	Connector symbol	Shows the directional flow, or control flow, of the activity. An incoming arrow starts a step of an activity; once the step is completed, the flow continues with the outgoing arrow.
	Joint symbol/ Synchronization bar	Combines two concurrent activities and re-introduces them to a flow where only one activity occurs at a time. Represented with a thick vertical or horizontal line.
	Fork symbol	Splits a single activity flow into two concurrent activities. Symbolized with multiple arrowed lines from a join.
	Decision symbol	Represents a decision and always has at least two paths branching out with condition text to allow users to view options. This symbol represents the branching or merging of various flows with the symbol acting as a frame or container.

Symbol	Name	Description
	Note symbol	Allows the diagram creators or collaborators to communicate additional messages that don't fit within the diagram itself. Leave notes for added clarity and specification.
	Send signal symbol	Indicates that a signal is being sent to a receiving activity.
	Receive signal symbol	Demonstrates the acceptance of an event. After the event is received, the flow that comes from this action is completed.
	Shallow history pseudostate symbol	Represents a transition that invokes the last active state.
	Option loop symbol	Allows the creator to model a repetitive sequence within the option loop symbol.
	Flow final symbol	Represents the end of a specific process flow. This symbol shouldn't represent the end of all flows in an activity; in that instance, you would use the end symbol. The flow final symbol should be placed at the end of a process in a single activity flow.
[Condition]	Condition text	Placed next to a decision marker to let you know under what condition an activity flow should split off in that direction.
	End symbol	Marks the end state of an activity and represents the completion of all flows of a process.


User
Home
FAQ
Contact Us
Forget Password
Register
Login
Creditionals are Incorrect!
OTP incorrect
Verify OTP
OTP incorrect
Verify OTP
Enter Creditionals
Creditionals are Correct
OTP is correct
OTP is correct User Added
System
Check User Type
New Password
Password updated!
Professor
Professor Dashboard
Student
Create Exam
Generate Questions
View Questions
Update Questions
Delete Questions
Test History
Student Results
Insert Marks
Publish Results
Student Monitoring
Change Password
Report Problems
Payment
Live Monitoring
Logout
Logout
Report Problems
Change Password
Results
Exam History
Student Dashboard
Give Exam
Exam Login
Exam Creditionals are incorrect!
Exam Login Creditionals are correct!
Exam
Proctoring

TUTORIAL NO: 09 UML Diagrams: State Diagram

THEORY:

What is a state diagram in UML?

A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives. States refer to the different combinations of information that an object can hold, not how the object behaves. In order to understand the different states of an object, you might want to visualize all of the possible states and show how an object gets to each state, and you can do so with a UML state diagram.

Each state diagram typically begins with a dark circle that indicates the initial state and ends with a bordered circle that denotes the final state. However, despite having clear start and end points, state diagrams are not necessarily the best tool for capturing an overall progression of events. Rather, they illustrate specific kinds of behavior—in particular, shifts from one state to another.

State diagrams mainly depict states and transitions. States are represented with rectangles with rounded corners that are labeled with the name of the state. Transitions are marked with arrows that flow from one state to another, showing how the states change. Below, you can see both these elements at work in a basic diagram for student life.

State diagram applications

Like most UML diagrams, state diagrams have several uses. The main applications are as follows:

- Depicting event-driven objects in a reactive system.
- Illustrating use case scenarios in a business context.
- Describing how an object moves through various states within its lifetime.
- Showing the overall behavior of a state machine or the behavior of a related set of state machines.

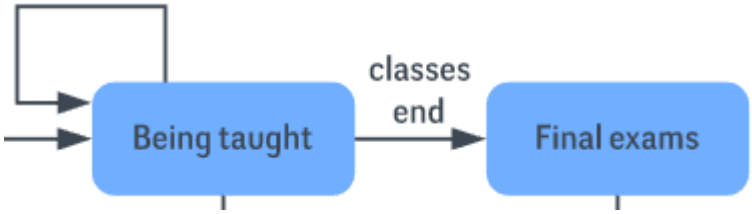
State diagram symbols and components

You can include many different shapes in a state diagram, particularly if you choose to combine it with another diagram. This list summarizes the most common shapes you may encounter.

Composite state
A state that has substates nested into it.

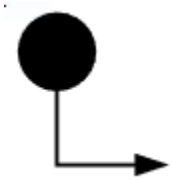
Choice pseudostate
A diamond symbol that indicates a dynamic condition with branched potential results.

Event
An instance that triggers a transition, labeled above the applicable transition arrow. In this case, “classes end” is the event that triggers the end of the “Being taught” state and the beginning of the “Final exams” state.



Exit point
The point at which an object escapes the composite state or state machine, denoted by a circle with an X through it. The exit point is typically used if the process is not completed but has to be escaped for some error or other issue.

First state
A marker for the first state in the process, shown by a dark circle with a transition arrow.



Guard
A Boolean condition that allows or stops a transition, written above the transition arrow.

State
A rectangle with rounded corners that indicates the current nature of an object.

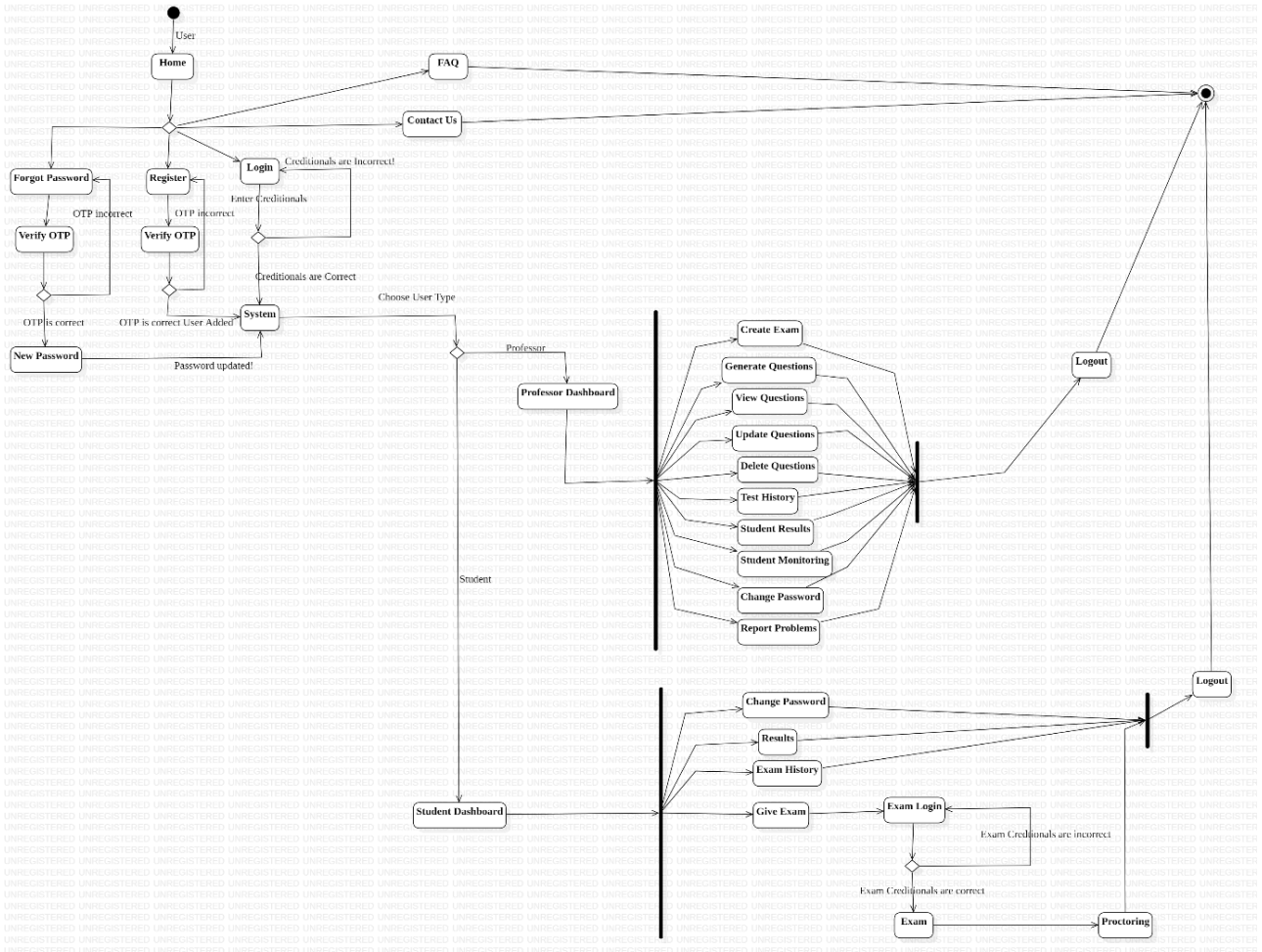
Substate
A state contained within a composite state's region.

Terminator

A circle with a dot in it that indicates that a process is terminated.

Transition

An arrow running from one state to another that indicates a changing state.



TUTORIAL NO: 10 UML Diagrams: Sequence Diagram

THEORY:

What is a sequence diagram in UML?

To understand what a sequence diagram is, it's important to know the role of the Unified Modeling Language, better known as UML. UML is a modeling toolkit that guides the creation and notation of many types of diagrams, including behavior diagrams, interaction diagrams, and structure diagrams.

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

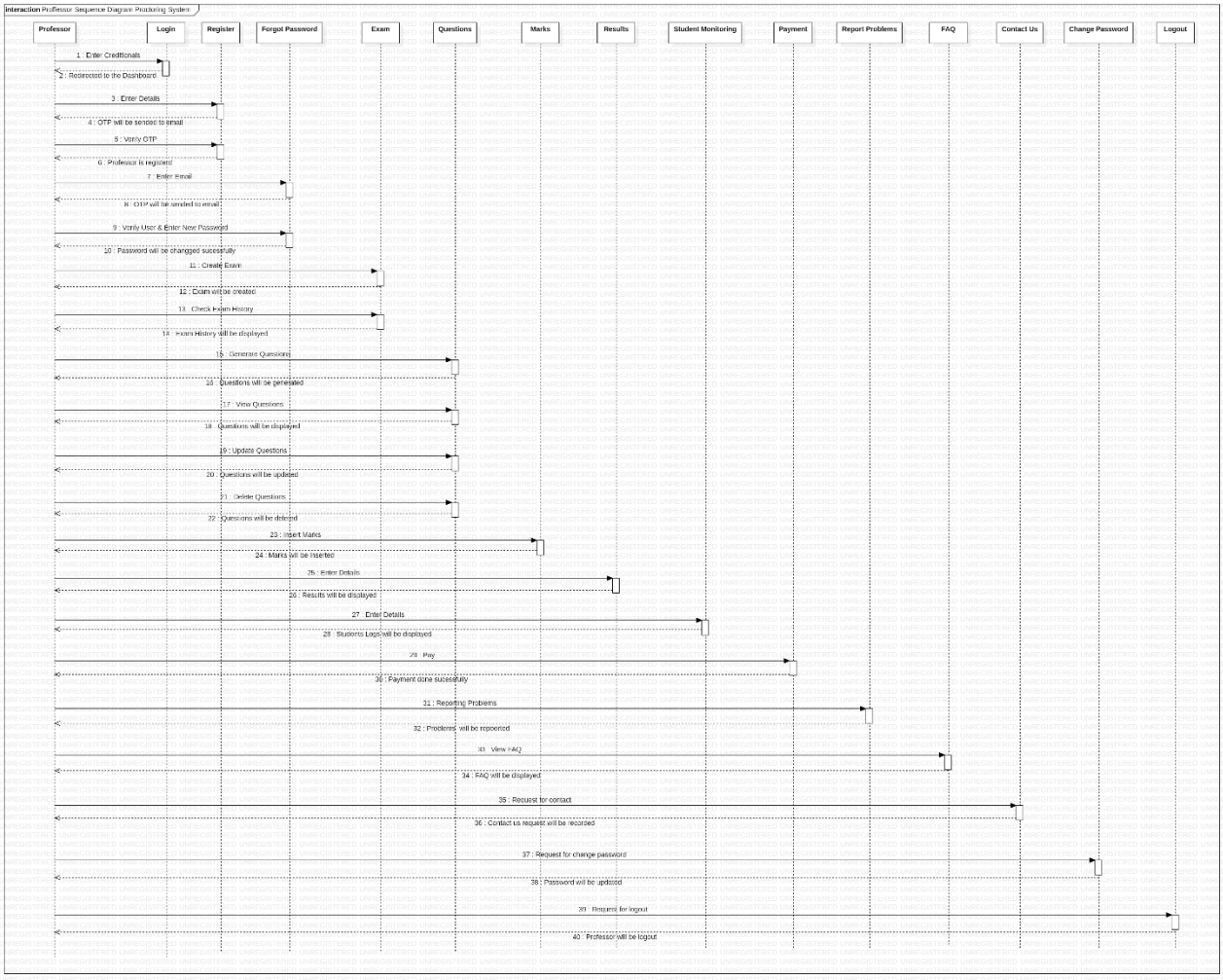
Note that there are two types of sequence diagrams: UML diagrams and code-based diagrams. The latter is sourced from programming code and will not be covered in this guide.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

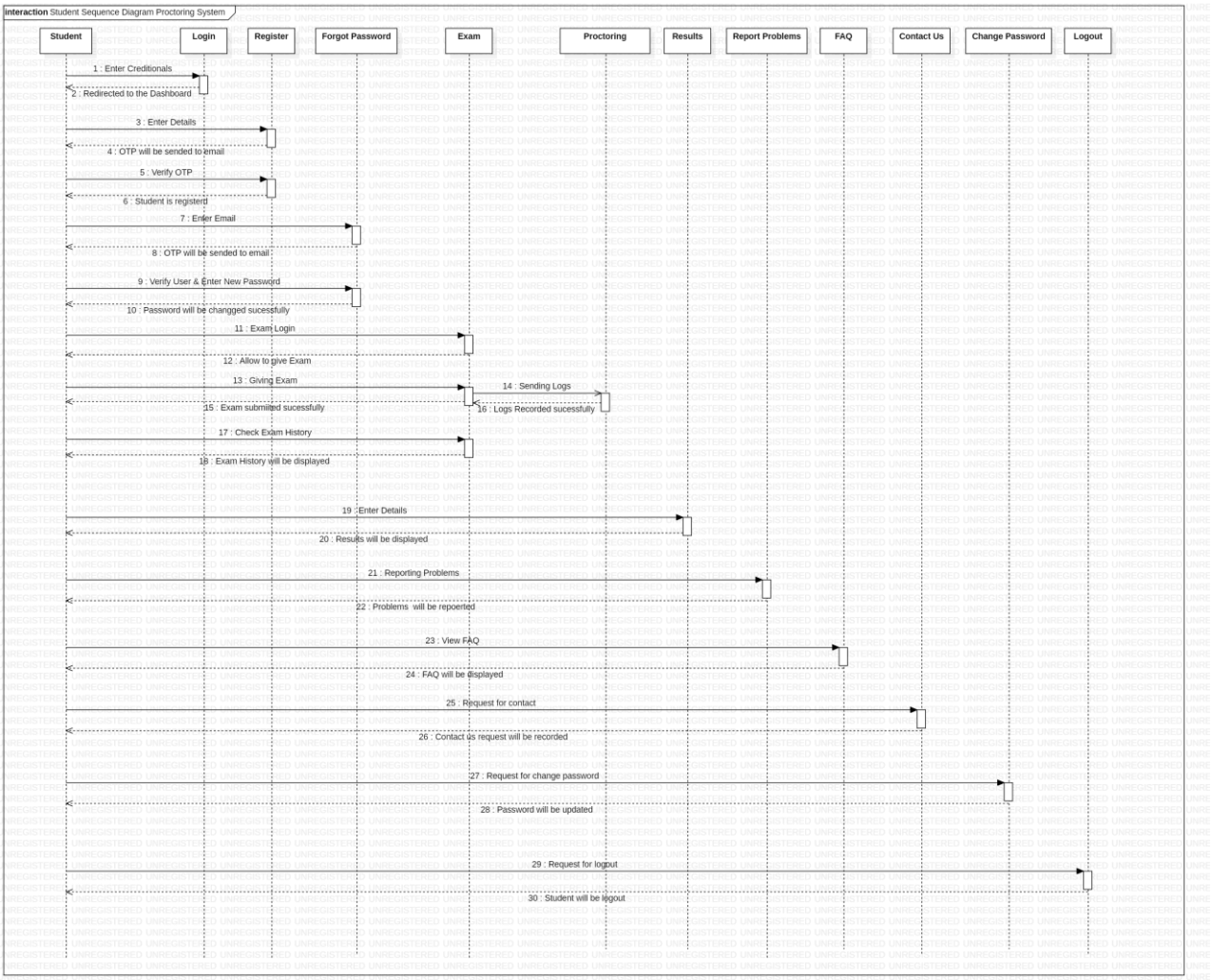
- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

PROFESSOR


Interaction Professor Sequence Diagram Proctoring System
Professor
Login
Register
Forgot Password
Exam
Questions
Marks
Results
Student Monitoring
Payment
Report Problems
FAQ
Contact Us
Change Password
Logout
1 : Enter Creditionals
2 : Redirected to the Dashboard
3 : Enter Details
4 : OTP will be sended to email
5 : Verify OTP
6 : Professor is registerd
7 : Enter Email
8 : OTP will be sended to email
9 : Verify User & Enter New Password
10 : Password will be changged successfully
11 : Create Exam
12 : Exam will be created
13 : Check Exam History
14 : Exam History will be displayed
15 : Generate Questions
16 : Questions will be generated
17 : View Questions
18 : Questions will be displayed
19 : Update Questions
20 : Questions will be updated
21 : Delete Questions
22 : Questions will be deleted
23 : Insert Marks
24 : Marks will be inserted
25 : Enter Details
26 : Results will be displayed
27 : Enter Details
28 : Students Logs will be displayed
29 : Pay
30 : Payment done sucessfully
31 : Reporting Problems
32 : Problems will be reposend
33 : View FAQ
34 : FAQ will be displayed
35 : Request for contact
36 : Contact us request will be recorded
37 : Request for change password
38 : Password will be updated
39 : Request for logout
40 : Professor will be logout

TUTORIAL NO: 10 UML Diagrams: Sequence Diagram

STUDENT



TUTORIAL NO: 11 UML Diagrams: Component Diagram

THEORY:

What is a UML component diagram?

The purpose of a component diagram is to show the relationship between different components in a system. For the purpose of UML 2.0, the term "component" refers to a module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.

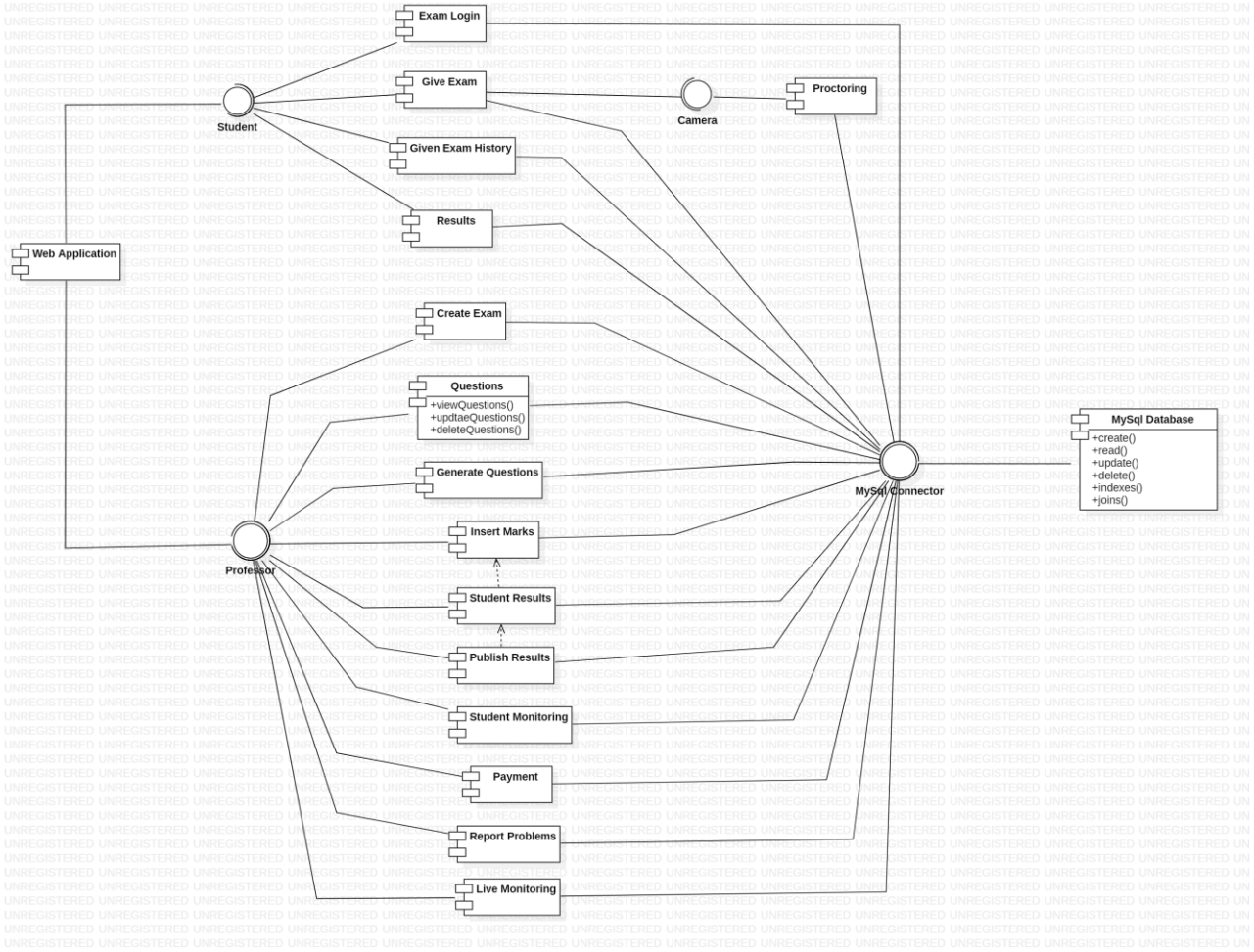
There exists a whole development approach that revolves around components: component-based development (CBD). In this approach, component diagrams allow the planner to identify the different components so the whole system does what it's supposed to do.

More commonly, in an OO programming approach, the component diagram allows a senior developer to group classes together based on common purpose so that the developer and others can look at a software development project at a high level.

Benefits of component diagrams

Though component diagrams may seem complex at first glance, they are invaluable when it comes to building your system. Component diagrams can help your team:

- Imagine the system’s physical structure.
- Pay attention to the system’s components and how they relate.
- Emphasize the service behavior as it relates to the interface.


Exam Login
Give Exam
Given Exam History
Results
Student
Camera
Proctoring
Web Application
Create Exam
Questions
+viewQuestions()
+updtaeQuestions()
+deleteQuestions()
MySql Database
+create()
+read()
+update()
+delete()
+indexes()
+joins()
Generate Questions
MySql Connector
Insert Marks
Professor
Student Results
Publish Results
Student Monitoring
Payment
Report Problems
Live Monitoring

TUTORIAL NO: 12 UML Diagrams: Deployment Diagram

THEORY:

What is a deployment diagram?

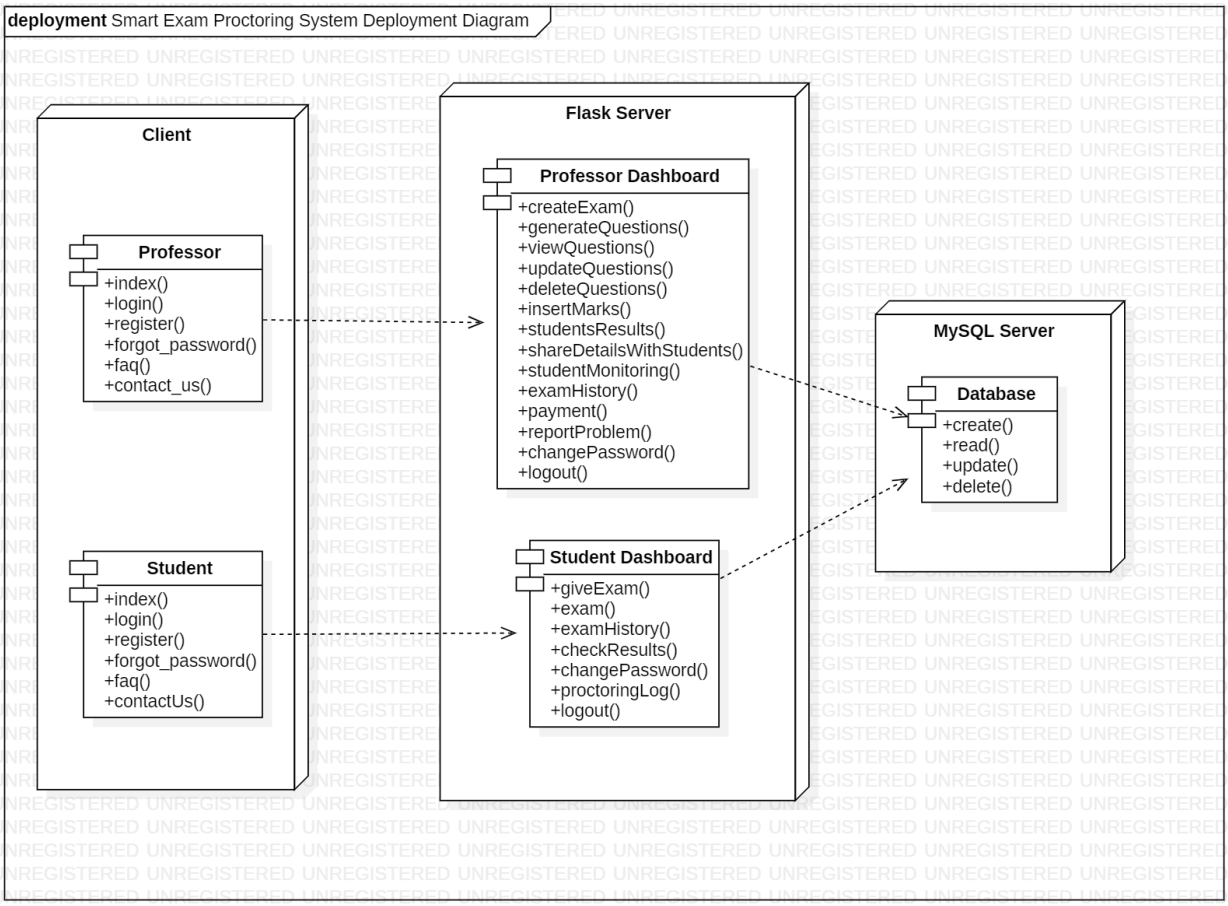
In the context of the Unified Modeling Language (UML), a deployment diagram falls under the structural diagramming family because it describes an aspect of the system itself. In this case, the deployment diagram describes the physical deployment of information generated by the software program on hardware components. The information that the software generates is called an artifact. This shouldn't be confused with the use of the term in other modeling approaches like BPMN.

Deployment diagrams are made up of several UML shapes. The three-dimensional boxes, known as nodes, represent the basic software or hardware elements, or nodes, in the system. Lines from node to node indicate relationships, and the smaller shapes contained within the boxes represent the software artifacts that are deployed.

Deployment diagram applications

Deployment diagrams have several valuable applications. You can use them to:

- Show which software elements are deployed by which hardware elements.
- Illustrate the runtime processing for hardware.
- Provide a view of the hardware system’s topology.


deployment Smart Exam Proctoring System Deployment Diagram
Client
Professor
+index()
+login()
+register()
+forgot_password()
+faq()
+contact_us()
Student
+index()
+login()
+register()
+forgot_password()
+faq()
+contactUs()
Flask Server
Professor Dashboard
+createExam()
+generateQuestions()
+viewQuestions()
+updateQuestions()
+deleteQuestions()
+insertMarks()
+studentsResults()
+shareDetailsWithStudents()
+studentMonitoring()
+examHistory()
+payment()
+reportProblem()
+changePassword()
+logout()
Student Dashboard
+giveExam()
+exam()
+examHistory()
+checkResults()
+changePassword()
+proctoringLog()
+logout()
MySQL Server
Database
+create()
+read()
+update()
+delete()