SRI SHAKTHI

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

21CS612 – OBJECT ORIENTED ANALYSIS AND DESIGN LABORATORY

LABORATORY RECORD

NAME: **ROLLNO:**

CLASS: **BRANCH:**

ACADEMIC YEAR: **BATCH:** **SEMESTER:**

Certified and Bonafide record of work done by _____

Place: Coimbatore

Date:

Staff In-Charge **Head of the Department**

University Register Number:

Submitted for the University Practical Examination held on……..……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Project Title: Online Course Management System

INDEX

Table of Contents

SOFTWARE REQUIREMENT SPECIFICATION

PROBLEM DEFINITION

An online course management system has to be developed. The system should contain the following features:

1. The system should contain a collection of courses which encompass practice tests, course material and assessments
2. A feature which displays the percentage of completion in enrolled courses
3. A Dashboard for students and the administrators. The administrators can view the progress of all students enrolled in the course. The students can check their own progress
4. The system must be embedded with an online compiler, essay writing space with word count and easy framing of MCQ questions for the tests and assessments
5. The system will have a login portal which will redirect the students and administrators to their respective dashboards
6. The Courses Page will display a list of courses enrolled and their completion statuses
7. The registration page will require the user's Email-Id, Date of Birth, Register Number and Contact Number

1. INTRODUCTION

Online Course Management System has been created with the aim of providing the students and faculty an easier and convenient way to impart knowledge. The functions of this system are done partly manually. Automated functions too play a role in completing the structure of the system.

The main modules of this system include:

- Student Module
- Administrator / Faculty Module

The system allows students to register and enrol for courses. They can attempt quizzes, look over course material that have been uploaded by faculties and look at their progress and test scores in the dashboard.
The system permits the faculty to create tests for students. They can also check the score of every student and their progress in their enrolled courses in the dashboard. They can upload various videos, presentations, documents in multiple formats as course or study material in the courses.

1.1 About Project

The project named "Online Course Management System" is built in a user – friendly nature. The built – in documentation, error detection, complication avoidance methods enable the system to act as a powerful tool in most cases.

The user will be working with a User Interface that's easy to navigate around in case they have any difficulties in doing so. The system has been created with utmost perfection so that there are no hassles whilst working on the software. Errors and bugs are rectified and reports are generated to users to guide them through the software.

Objectives of the Project:

- Help students enrol in courses
- Provide administrators to create tests and upload course material with ease

1.2 Purpose

The purpose of this document is to present a detailed description of the course management system. This document will explain the features, their working and how each feature of the system is called. This document will prove to be helpful to both stakeholders and developers.

The purpose of the system is to provide a user – friendly portal to students and encourage them to enrol in courses based on their interests and inclinations. It will help them focus on a particular branch and develop their skills in the same.

1.3 Scope of the Project

The course management system is built on the idea to provide students the information and tools required to develop their skills in a particular field.

The system will be of tremendous help to students because of the easy and friendly navigation around the system. The system will also keep a record of the progress so that the user will not have to start afresh every time they log in.

Need for Computerisation

- Response Time
- Recovery Time
- Throughput
- Start – up Time
- Capacity

1.4 Project Overview

Existing System:

Need for an online course management system have gone up in these trying times. However, the efficiency of these systems seems to be hit because of the concurrency of the servers of certain course management systems.

Only on analysis, the flaws and errors in these existing systems can be concluded. The aim of the system being built is to avoid these errors and flaws and provide a clean and fast software to the students and users. A good analysis model should provide not only the mechanisms of problem understanding but also the frame work of the solution. Thus, it should be studied thoroughly by collecting data about the system. Then the proposed system should be analysed thoroughly in accordance with the needs.

System analysis can be categorized into four parts.

- System planning and initial investigation
- Information Gathering
- Applying analysis tools for structured analysis
- Feasibility study
- Cost/ Benefit analysis.

Proposed System:

In the proposed system, the test creation and material uploading mechanisms will be done a lot faster and quicker because of cloud servers and the all students will be able to access it without much hassle. The grade calculation process is automated and it can generate the results as soon as the student is done with their test and submits it.

1.5 REFERENCES

https://www.slideshare.net/marwa_alamri/srs-document-of-course-management-software-systemdoc

https://www.ques10.com/p/10355/prepare-srs-for-the-course-management-system-1/

2. OVERALL DESCRIPTION

2.1 Product Perspective

Online course management system serves the purpose of making online education easier and more personalized than the general classroom education. The product is designed keeping in mind the usual pain points that, students tend to report against other online education systems available and strives to make the experience more convenient. The User Interface is very intuitive and has almost no learning curve hence helping both the student and the staff save a lot of time and effort. The significant advantage the online course management system has over the traditional methods is that the student can go at their own pace as long as they could finish the given task under the deadline. This allows the student to grasp the concepts better and spend more time on the concepts that they feel are a little tough. The staffs also have it easier this way in terms of tracking the students individually and make their guidance more personalized. With the online course management system, the staff will be able to upload the study materials and tests of their choice and also individually monitor every student's progress. The students can complete the tasks in their own time and also get an analysis of their performance. It also provides the student with an easier way of keeping track of their overall performance with little to no effort.

2.2 Product Features

1. Home page

In home page the student can view the courses that they have enrolled in and the gross completion percentage as modules or blocks arranged side by side. The admin can view the courses that they have put up in the place of the blocks. The navigation bar can have links to take them to the other pages.

2. Student Registration

In student registration, the student needs to fill the following details:

I. Student Name
II. Student Register Number
III. Student Email ID
IV. Student Mobile Number
V. Date of Birth
VI. Password

3. Student Login

By using their E-mail id and password student can login to access the site.

4. Viewing Student Details

a. The student can view their details which includes the following:

i. Register number

ii. Student name

iii. Their self-written bio

iv. Department details

v. E-mail id

b. The student can view the courses that they are currently enrolled in as well as the courses that they completed in the home.

c. The student can view the dashboard of a course for a detailed analysis of their performance

5. Admin Login

By using admin email id and password admin can access the site.

6. Faculty

The faculty will be taken to a page where they can access the courses that they created. The courses will be arranged as modules side by side in the same fashion as the student's home page

2.3 User Classes and Characteristics

- The intended users of this software need not have specific knowledge as to what is the internal operation of the system. Thus, the end user is at a high level of abstraction that allows easier, faster operations and reduces the knowledge requirement of end user.
- The product is absolutely user friendly and intuitive, so the intended users can be the naive users.
- The system does not expect the end users to have any technical expertise for navigating through the software.

2.4 Operating Environment

The web application is hosted in any cloud platform or local server to be accessed by any browser.

2.5 Design and Implementation Constraints

- In home page anyone can view the college overall information and it contains a registration and login button which is available for student to login and they can view their details.
- In home page of the student, they can see the courses that they have enrolled in as well as the navigation bar that can take them to the other pages of the site.
- Upon entering a course, they will find the course with two parts.
- One part will be the course content along with progress tracking and the other part, also called the dashboard, is for displaying the analysis of performance after they complete the course.
- The student can use the link in the navigator bar to go to their details page where their login details such as their name, register number, phone and email will be displayed for their perusal as well as updating.
- By using admin id and password admin can login to a home page customized to their use case.
- They can access the courses that they are authoring as well as the progress of the students in the respective courses.
- Faculty use the same link as the one in the student home page to view their login credentials as well the student credentials.

2.6 User Documentation

User manual is available for all type of users in the website ready.

2.7 Assumptions and dependencies

- Login enabled by college E-mail id for students.
- Faculty can login by using their staff email id.

3. SYSTEM FEATURES

3.1 Login/sign up

1. Description and priority

a. Login is done by the student & admin in the same login page
b. It is a high priority feature as the college website shouldn't allow user to go forward without this step.

2. Response Sequence

a. Login/sign up action is done by the student & admin.
b. The system's response time will be less than minutes and the user won't have to wait for long.

3. Functional Requirements

User satisfaction: the system is designed in such a way that the user can start using the site intuitively without any need for guidance.

Error handling:

If it is a new user account or email, the system should create an account by after making sure that the email hasn't been registered with the site already.
If the user mail or credentials are already there in the database, then the system should move to forget password or user already exist prompt.
If they have forgotten the password, then forgot password feature comes up and processes.

3.2 Access contents of the site

I. Description and priority

- The student will be able to access the course modules first thing after they enter the site and analysing their performance will take the place of the next priority
- These two are high priority features

II. Functional Requirements

- The student can access their course materials and learning progress using the sophisticated data analysis tools of the system in the dashboard.

3.2.1 Modify search

If the user has given the information incorrectly, then using the features such as update, display, search, they can change it.

3.3 Access course contents

The student can enter a course module by clicking on it. Inside the course they have access to two functionalities. The first one will be around the course content which they can use in their own time. The other functionality will be a dashboard to track their progress after they've completed a course.

Functional requirements

1. The function helps the student enter the course material where they left it during the last session.
2. The progress of student will be updated in real time and the student won't lose their progress if the system shuts down unexpectedly.

3. The user and the admin can track the progress of a student only after they've completed the course.
4. The user won't be able to access a course after the deadline (if mentioned)

3.4 Admin view

The admin can access and manipulate the courses that they are authoring as well as their personal details. They can view the student details without changing anything.

Functional requirements

1. This function allows the user to view the courses that they created and the progress of the student.
2. The function allows the progress of the staff to be retained if their system shuts down by mistake.
3. The user can view the details of their own as well as the students but can edit only theirs.

3.5 Dashboard

After the student has finished attending the course, their progress will to be updated to the system. A student needs to complete their module for the admin to have access to the dashboard too.

Functional requirements

a. Access

i. The student can access the dashboard located inside a module for the analytics of that particular course.

b. Analytics

ii. The analysis will be done using some of the most sophisticated data analytics and visualization tools available.

3.6 Course completion

After the authoring a course, the progress will be saved and will be available for the students to access.

4. External Interface Requirements

4.1 Hardware interfaces

1. **Hard disk:** The database connectivity requires a hardware configuration with a fast database system running on high rpm hard-disk permitting complete data redundancy and back-up systems to support the primary goal of reliability.

2. The system must interface with the standard output devices, keyboard and mouse to interact with this software.

4.2 Software interfaces

- ✓ Back end: TBD
- ✓ Front end: TBD
- ✓ Database: TBD

4.3 Operations

The admin mode enables the end-users to do the end user operations like viewing updated schedules and upcoming exams.

4.4 Communication interfaces

TBD

5. Other Non-Functional Requirements

5.1 Performance Requirements

This software should be able to handle the following tasks:

1. An Average rate of response time shall be less than 2 seconds.
2. The System shall accommodate 100 students consecutively.
3. The Average repair time of the system in case of a system failure shall be less than half hour.
4. The Storage Capacity of the system shall be at least 1000.

5.2 Safety Requirements

1. The system shall be able to handle the load for courses, tests, videos and PPT.
2. The system shall be able to handle the large number of students.
3. The system shall not be collapsed of handling a certain number of students consecutively.
4. The system shall not be down more than 2 times in a year.

5.3 Security Requirements

1. Online course management system shall run inside a firewall
2. The system shall support different roles for students, Faculty and Administrative staff
3. The students shall only be allowed access consistent with that role.

5.4 Software Quality Attributes

1. The Online course management system shall possess well organized student's dashboard

2. The system shall enable the students to learn at better pace like providing tests, videos etc.,

6. Other Requirements

NA

Use Case Diagram

Definition:

To create use case diagram for Online Course Management system using a computer aided software engineering tool - Creately

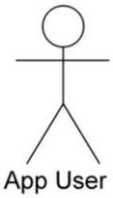
Explanation:

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

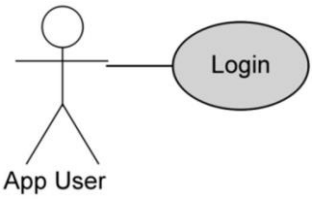
Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. The use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

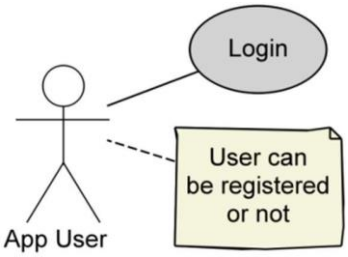
Actor



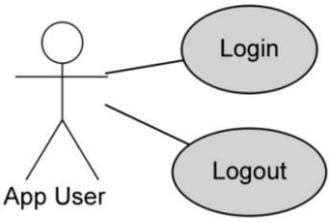
Actor and Use Case



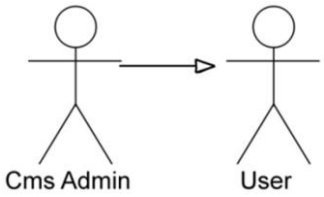
Notes



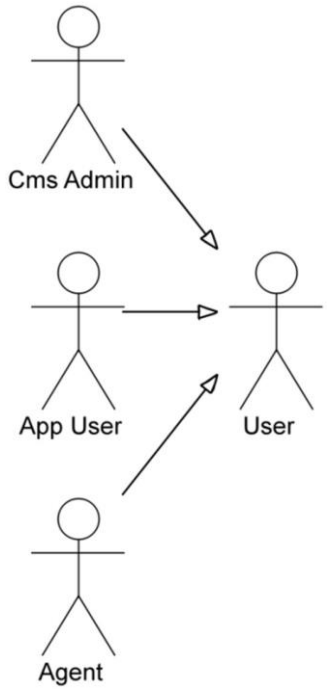
Many Use Cases



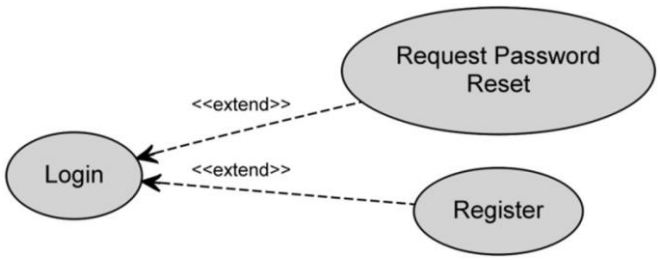
Actor Inheritance



Multiple Actors and Inheritance



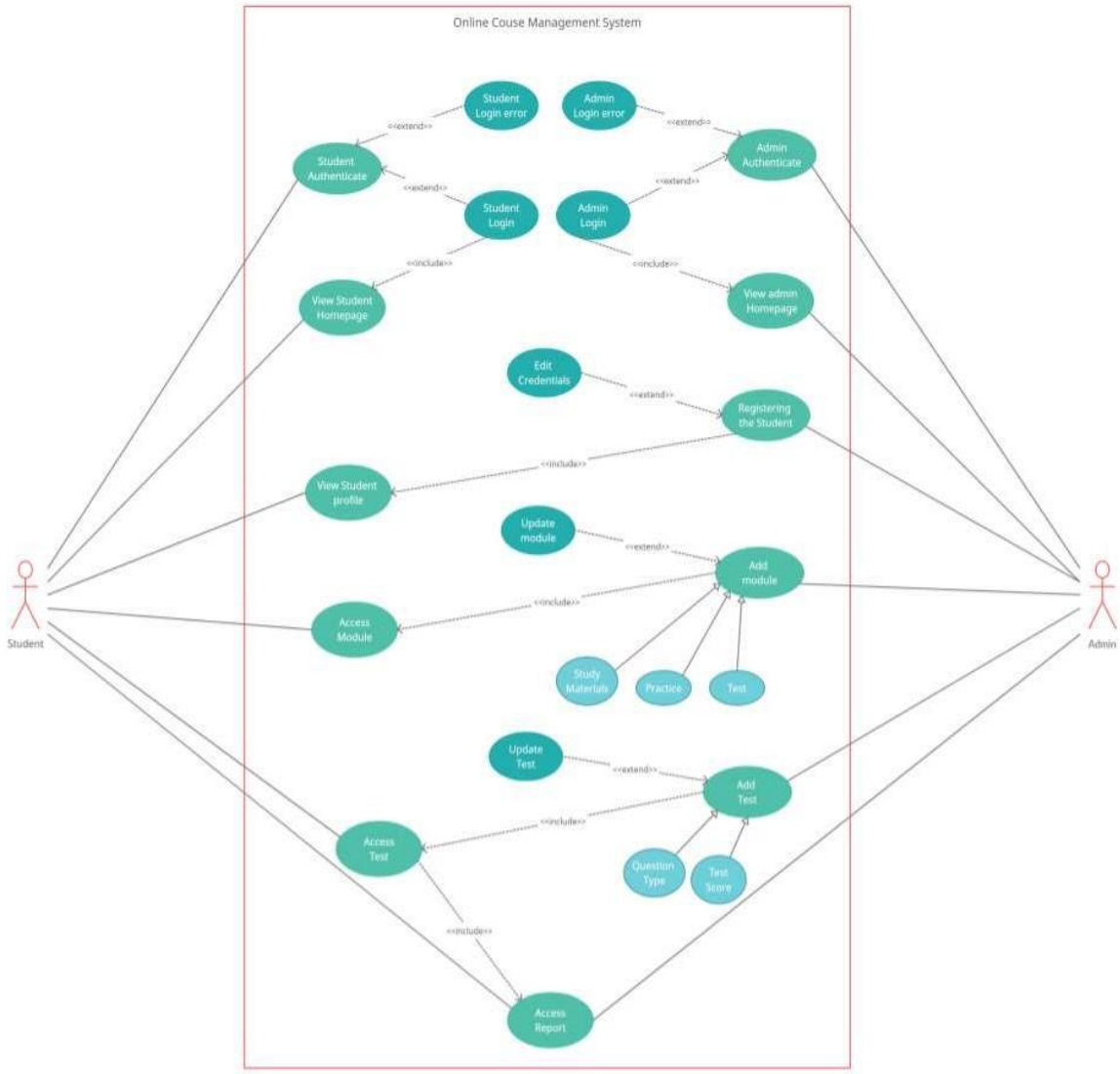
<<Extends>>



<<Includes>>

Use Case Diagram:

Description:



A use case diagram is a graphical depiction of a user's possible interactions with a system. The actors of the Online Course Management System are Admin and Student. The admin's roles involve registering a Student, authorization, creating, updating and editing modules and exams. The roles of the Student involve accessing homepage, modules, exams and the scores. The admin authorization extends admin login and error. The Student authorization extends Student login and error. Registering a module and a exam module both extend updating them respectively. Student and Admin login include providing access to the Student to view their respective homepages. Registering a Student includes providing access to profile. Adding a

module includes providing access to the module to the student. Adding an exam module includes providing access to the module to the Student.

UML Class Diagram

Definition:

To develop the UML Class Diagram for the project Online Course Management System using the online Creately - Computer aided software engineering.

Explanation:

A class diagram is a UML diagram type that describes a system by visualizing the different types of objects within a system and the kinds of static relationships that exist among them. It also illustrates the operations and attributes of the classes.

They are usually used to explore domain concepts, understand software requirements and describe detailed designs.

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system. class diagrams are used for –

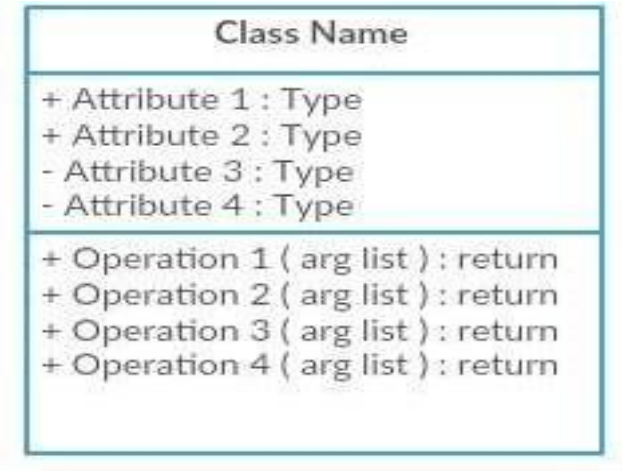
- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object-oriented languages.

Class Diagram Notations with Examples

There are several class diagram notations that are used when drawing UML class diagrams.

We've listed below the most common class diagram notations.

Class



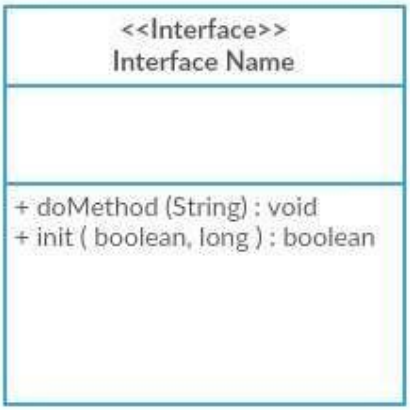
Classes represent the central objects in a system. It is represented by a rectangle with up to 3 compartments.

The first one shows the class's name, while the middle one shows the class's attributes which are the characteristics of the objects. The bottom one lists the class's operations, which represents the behavior of the class

Simple Class

The last two compartments are optional. The class notation without the last two compartments is called a simple class and it only contains the name of the class.

Interface



The interface symbol in class diagrams indicates a set of operations that would detail the responsibility of a class.

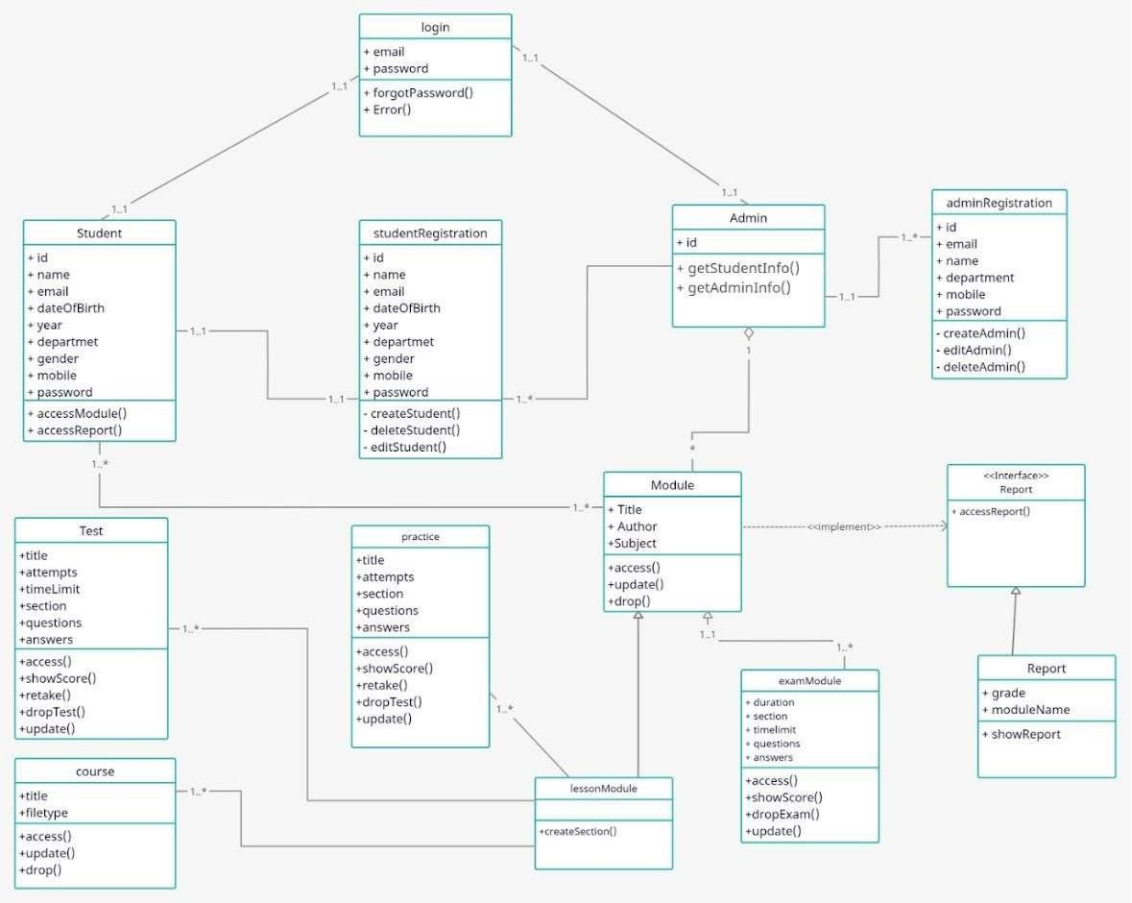
Package

The package symbol is used to group classes or interfaces that are either similar in nature or related. Grouping these design elements using the package symbols improves the readability of the diagram

Class Diagram Relationships

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

UML Class Diagram:



Description:

The studentRegistration and adminRegistration classes are used to create objects Student and Admin respectively. The login class is called by the Student and Admin objects at the time of login. The Admin object can create an object of the Module class and the Student object can access it. The module class has two subclasses representing two kinds of modules that can be created: lessonModule and examModule. The lessonModule in turn has three subclasses that represent the type of submodules that can be created under a lesson module: test, course and practice. An object of the above classes need to created for every module or submodule. The module class implements and interface called Report to generate analytics about the student progress. The report interface extends a class called Report which can be used to create objects for the respective module which can be later displayed as the report.

UML Sequence Diagram

Definition:

The Sequence diagram for an Online Course Management System

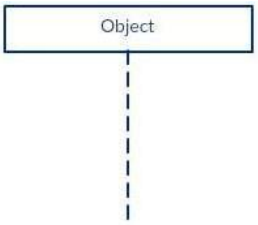
Procedure:

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

A Quick Overview of the Various Parts of a Sequence Diagram

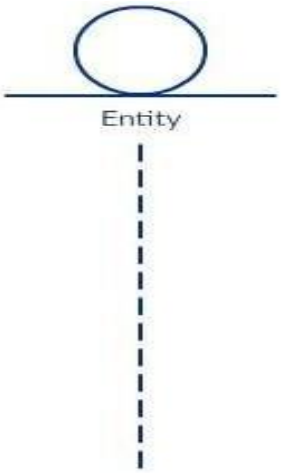
Lifeline Notation



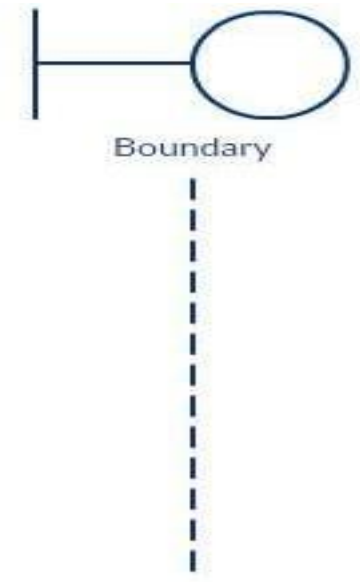
A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case .

A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer.



A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries.



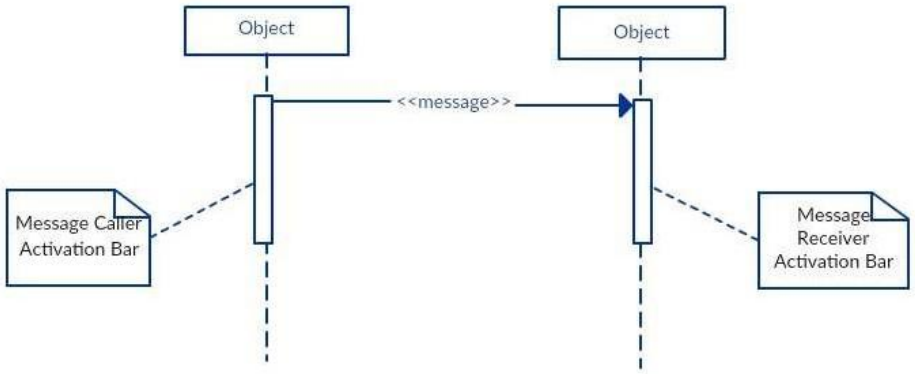
And a lifeline with a control element indicates a controlling entity or manager. It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them

Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the

object that receives the message) indicates that both are active/is instantiated during the exchange of the message



Message Arrows

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below. All parts except the message_name are optional.

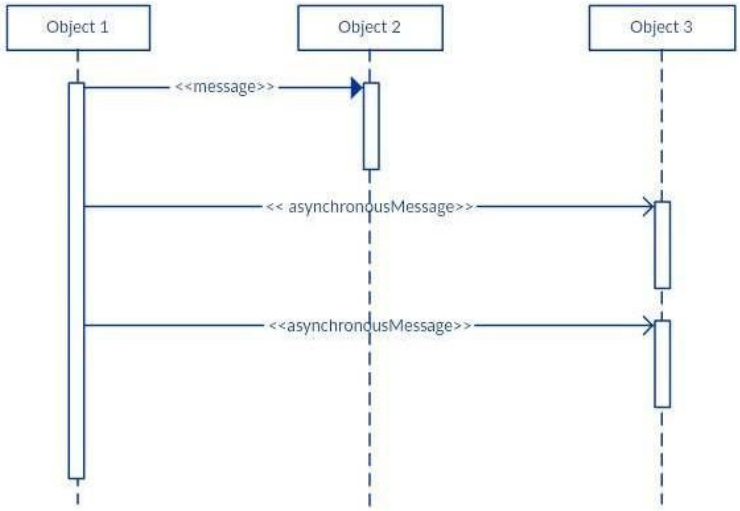
attribute = message_name (arguments): return_type

- **Synchronous message**

As shown in the activation bars example, a synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below

- **Asynchronous message**

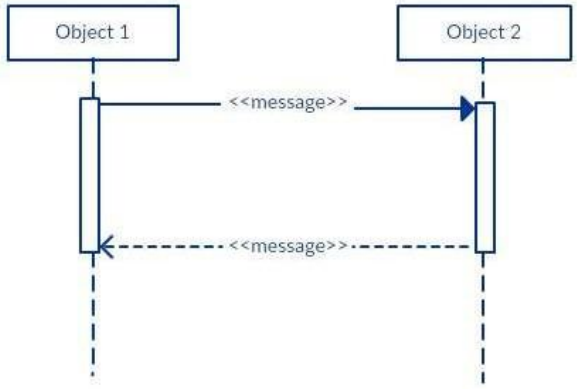
An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below



- **Return message**

A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller. Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

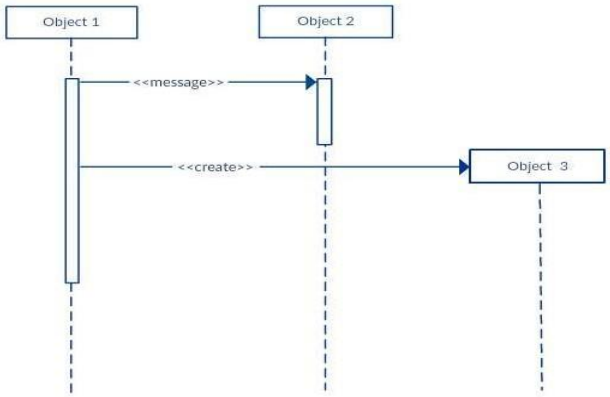
Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself



- **Participant Creation Message**

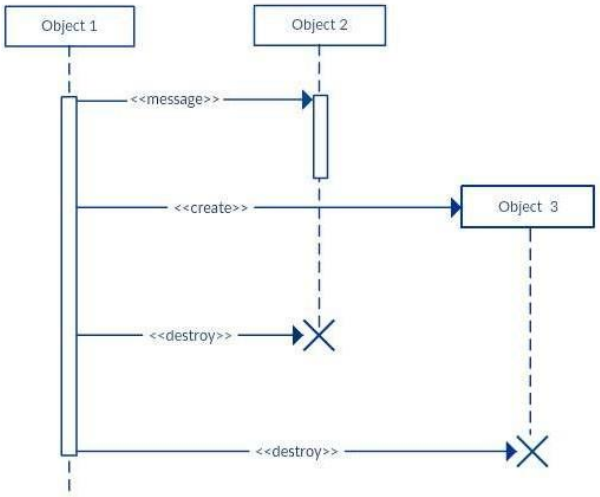
Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.

The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent. If the created participant does something immediately after its creation, you should add an activation box right below the participant box.



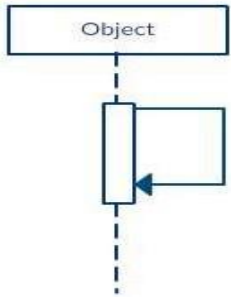
- **Participant destruction message**

Likewise, participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of the said participant.



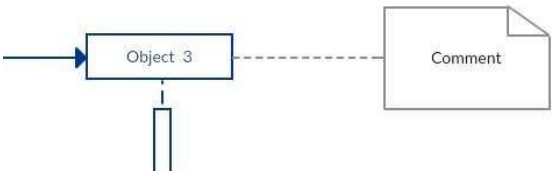
Reflexive message

When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below



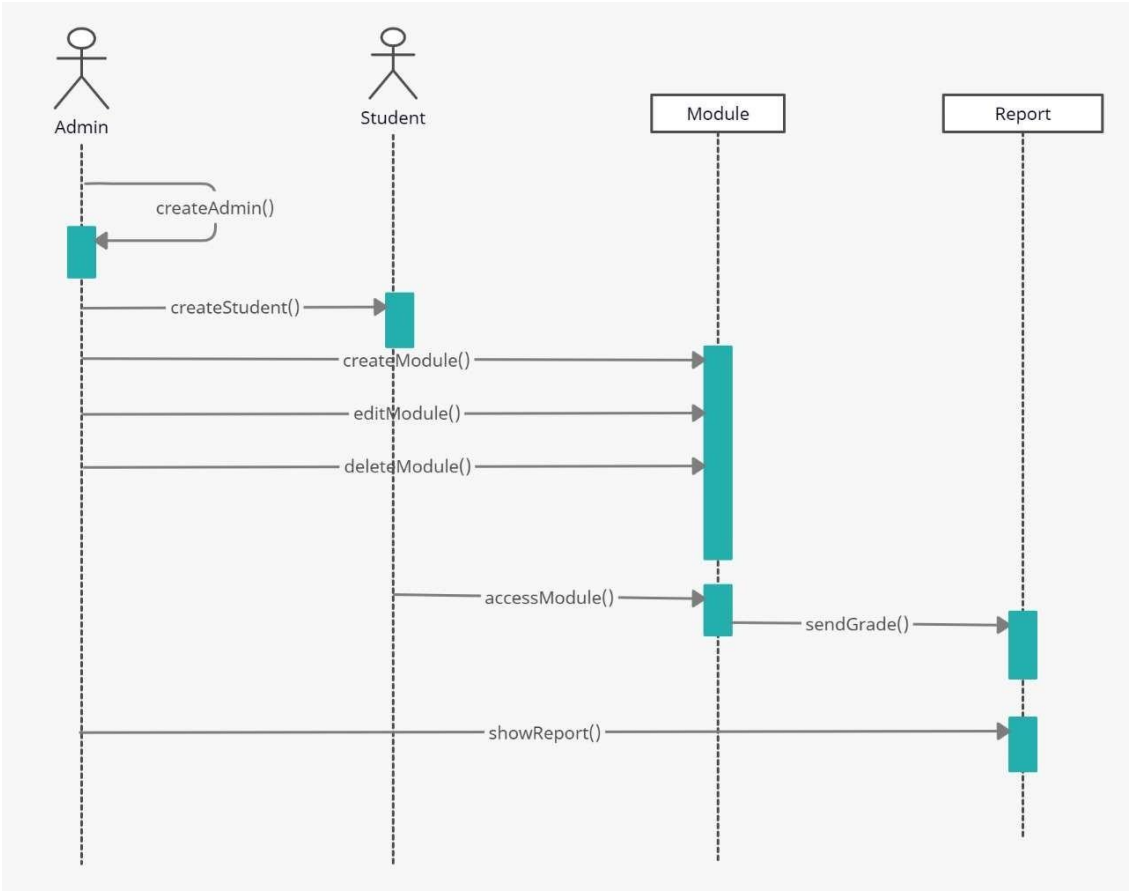
Comment

UML diagrams generally permit the annotation of comments in all UML diagram types. The comment object is a rectangle with a folded-over corner as shown below. The comment can be linked to the related object with a dashed line.



Note: View Sequence Diagram Best Practices to learn about sequence fragments

SEQUENCE DIAGRAM:



Description:

The actors are Student and Admin. The Admin calls the createAdmin method to create an Admin object and gets the return message as admin created. The Admin calls createStudent to create a Student object. The Admin invokes the createModule to create a module object. The Admin invokes the editModule method to edit the module and the deleteModule to delete the module object.

The Student calls the accessModule that involves the Module object that in turn directs to the Report object to fetch the results using sendGrade method. The Admin invokes the showReport method that directs to the Report object when it wants access the grades.

UML Collaboration Diagram

Definition:

To develop the UML collaboration diagram for the Online Course Management System using computer aided software engineering tool – creately

Procedure:

- A Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.
- A Collaboration defines a set of participants and relationships that are meaningful for a given set of purposes
- A Collaboration between objects working together provides emergent desirable functionalities in Object-Oriented systems
- Each object (responsibility) partially supports emergent functionalities
- Objects are able to produce (usable) high-level functionalities by working together
- Objects collaborate by communicating (passing messages) with one another in order to work together

Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways.

Because of the format of the collaboration diagram, they tend to better suited for analysis activities (see Activity: Use-Case Analysis). Specifically, they tend to be better suited to depicting simpler interactions of smaller numbers of objects. However, if the number of objects and messages grows, the diagram becomes increasingly hard to read. In addition, it is difficult to show additional descriptive information such as timing, decision points, or other unstructured information that can be easily added to the notes in a sequence diagram. So, here are some use cases that we want to create a collaboration diagram for:

- Model collaborations between objects or roles that deliver the functionalities of use cases and operations
- Model mechanisms within the architectural design of the system
- Capture interactions that show the messages passing between objects and roles within the collaboration
- Model alternative scenarios within use cases or operations that involve the collaboration of different objects and interactions
- Support the identification of objects (hence classes) that participate in use cases
- Each message in a collaboration diagram has a sequence number.

- The top-level message is numbered 1. Messages sent during the same call have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

Notations of Collaboration Diagram

Objects

An object is represented by an object symbol showing the name of the object and its class underlined, separated by a colon:

Object_name : class_name

You can use objects in collaboration diagrams in the following ways:

- Each object in the collaboration is named and has its class specified
- Not all classes need to appear
- There may be more than one object of a class
- An object's class can be unspecified. Normally you create a collaboration diagram with objects first and specify their classes later.
- The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.

Actors

Normally an actor instance occurs in the collaboration diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them in the periphery of the diagram.

- Each Actor is named and has a role
- One actor will be the initiator of the use case

Links

Links connect objects and actors and are instances of associations and each link corresponds to an association in the class diagram

Links are defined as follows:

- A link is a relationship among objects across which messages can be sent. In collaboration diagrams, a link is shown as a solid line between two objects.
- An object interacts with, or navigates to, other objects through its links to these objects.
- A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified.
- Message flows are attached to links, see Messages.

Messages

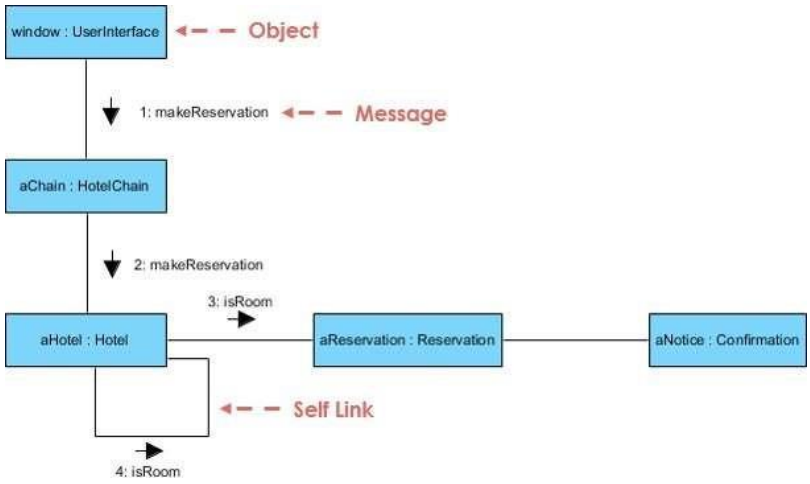
A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link.

- The message is directed from sender to receiver
- The receiver must understand the message
- The association must be navigable in that direction

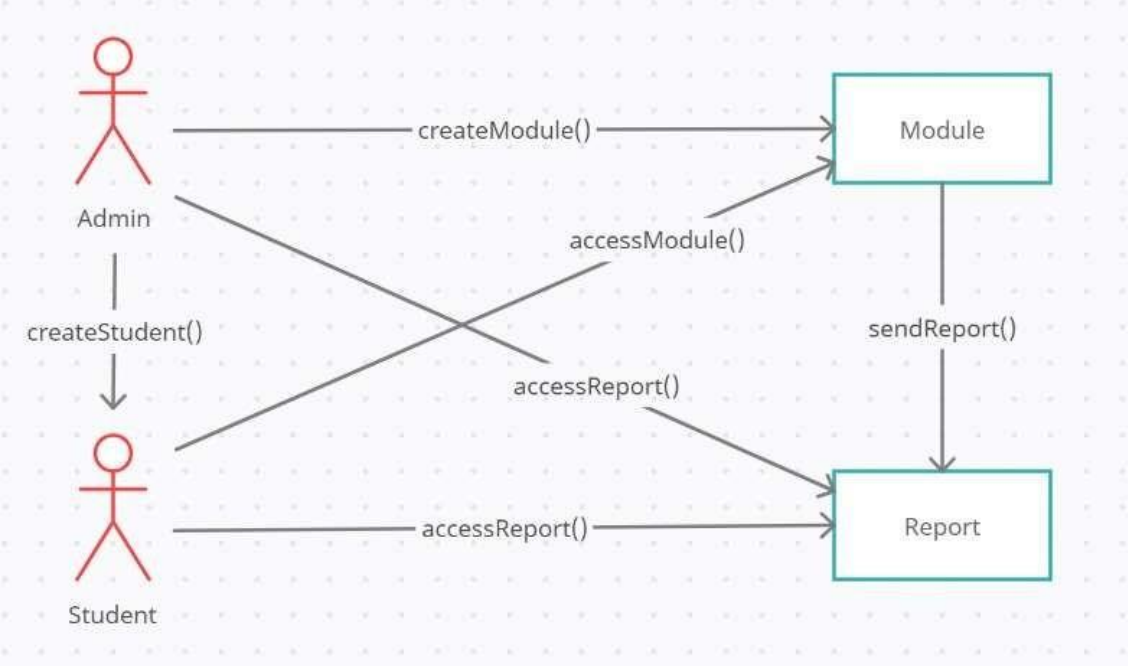
Steps for Creating Collaboration Diagrams

- Identify behavior whose realization and implementation is specified
- Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
- Decide on the context of interaction: system, subsystem, use case and operation
- Model structural relationships between those elements to produce a diagram showing the context of the interaction
- Consider the alternative scenarios that may be required
- Draw instance level collaboration diagrams, if required.
- Optionally draw a specification level collaboration diagram to summarize the alternative scenarios in the instance level sequence diagrams

Collaboration Diagram Example:



Collaboration Diagram:



Description:

The Admin actor invokes the createModule that creates a Module object. The Admin invokes the accessReport method to create The Admin can invoke the createStudent method to create a Student object. The Student can then access the module created using accessModule method. The module object invokes the sendReport method to create a Report object that can be accessed by the Student and the Admin by invoking their respective accessReport methods.

UML Activity Diagram

Definition:

To develop the Activity Diagram for the project Online Course Management System using the online Computer aided software engineering - tool creately

Procedure:

Activity Diagram Notations –

Initial State –
The starting state before an activity takes place is depicted using the initial state.

Figure – notation for initial state or start state

A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

For example – Here the initial state is the state of the system before the application is opened.

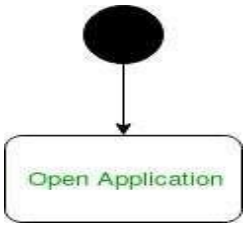


Figure – initial state symbol being used

Action or Activity State –
An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

Figure – notation for an activity state

For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

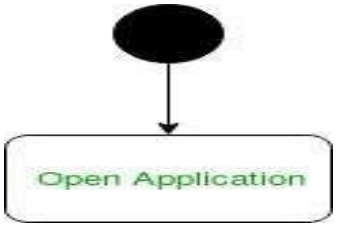


Figure – activity state symbol being used

Action Flow or Control flows –

Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

Figure – notation for control Flow

An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.

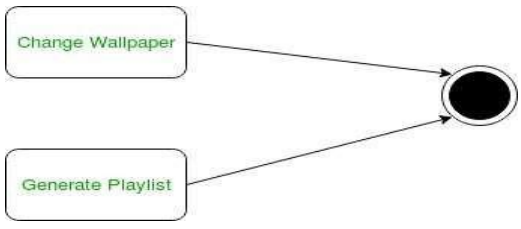


Figure – using action flows for transitions

Decision node and Branching –

When we need to make a decision before deciding the flow of control, we use the decision node.

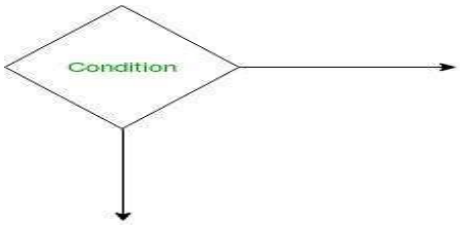


Figure – notation for decision node

The outgoing arrows from the decision node can be labelled with conditions or guard expressions.It always includes two or more output arrows.

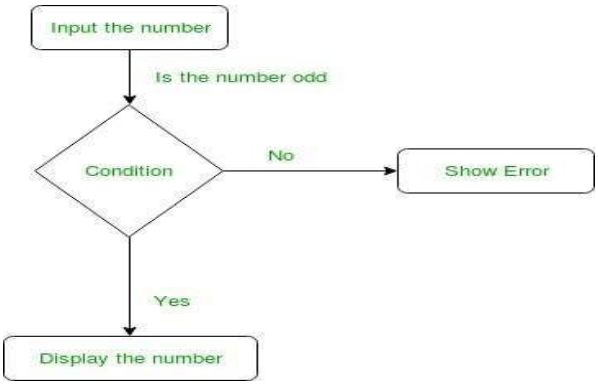


Figure – an activity diagram using decision node

Guards –

A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

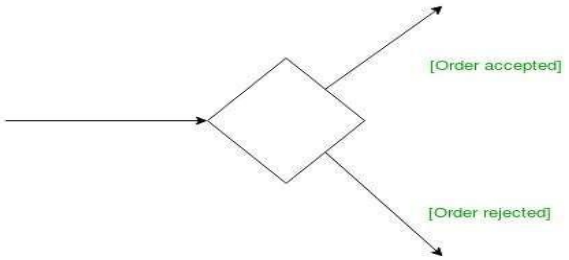


Figure – guards being used next to a decision node

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

Fork –

Fork nodes are used to support concurrent activities.

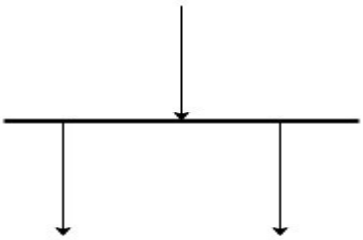


Figure – fork notation

When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.

For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.

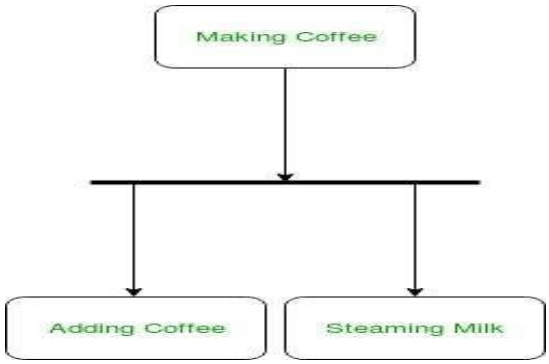


Figure – a diagram using fork

Join

Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

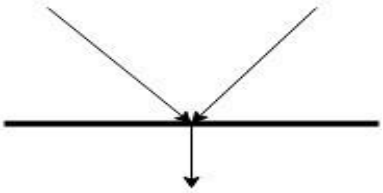


Figure – join notation

For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.

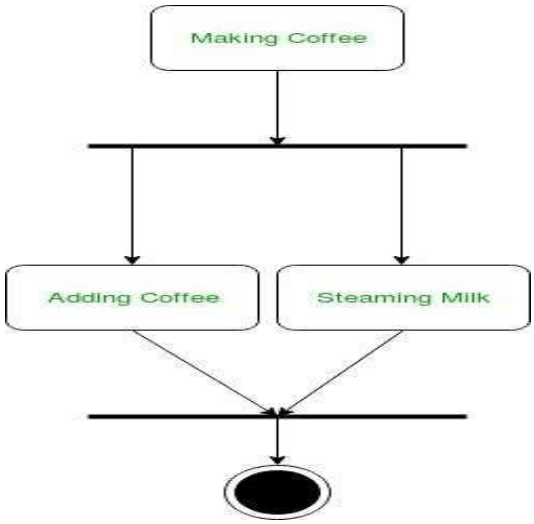


Figure – a diagram using join notation

Merge or Merge Event

Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

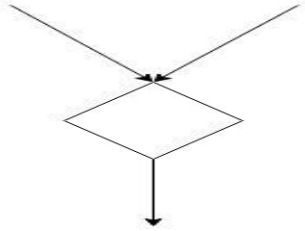


Figure – merge notation

For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time

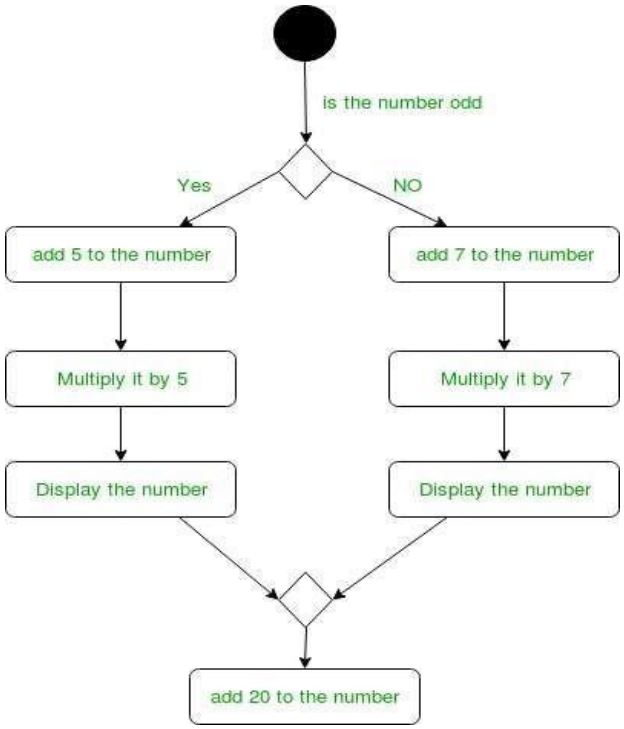


Figure – an activity diagram using merge notation

Swimlanes

We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

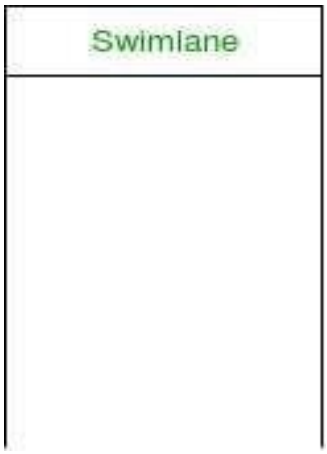


Figure – swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above.

For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.

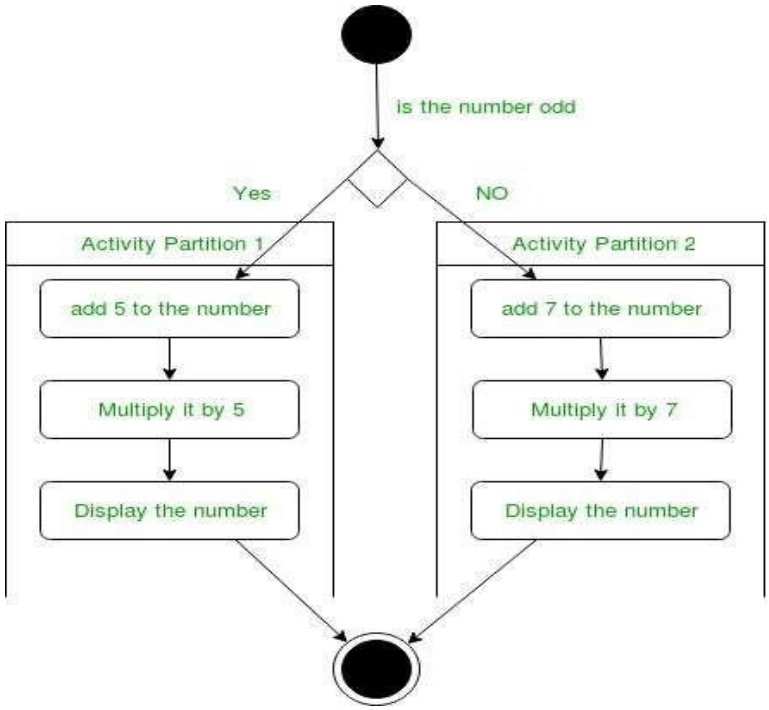


Figure – an activity diagram making use of swimlanes

Figure – time event notation

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.
For example – Let us assume that the processing of an image takes takes a lot of time. Then it can be represented as shown below.

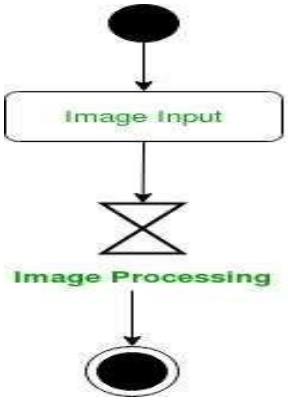


Figure – an activity diagram using time event

Final State or End State –

The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

Figure – notation for final state

How to Draw an activity diagram –

- Identify the initial state and the final states.
- Identify the intermediate activities needed to reach the final state from he initial state.
- Identify the conditions or constraints which cause the system to change control flow.
- Draw the diagram with appropriate notations.

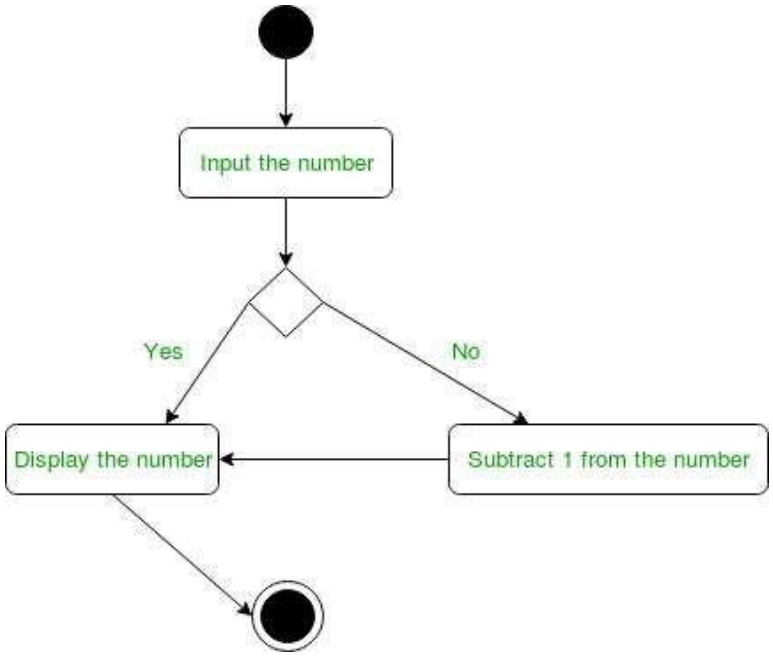
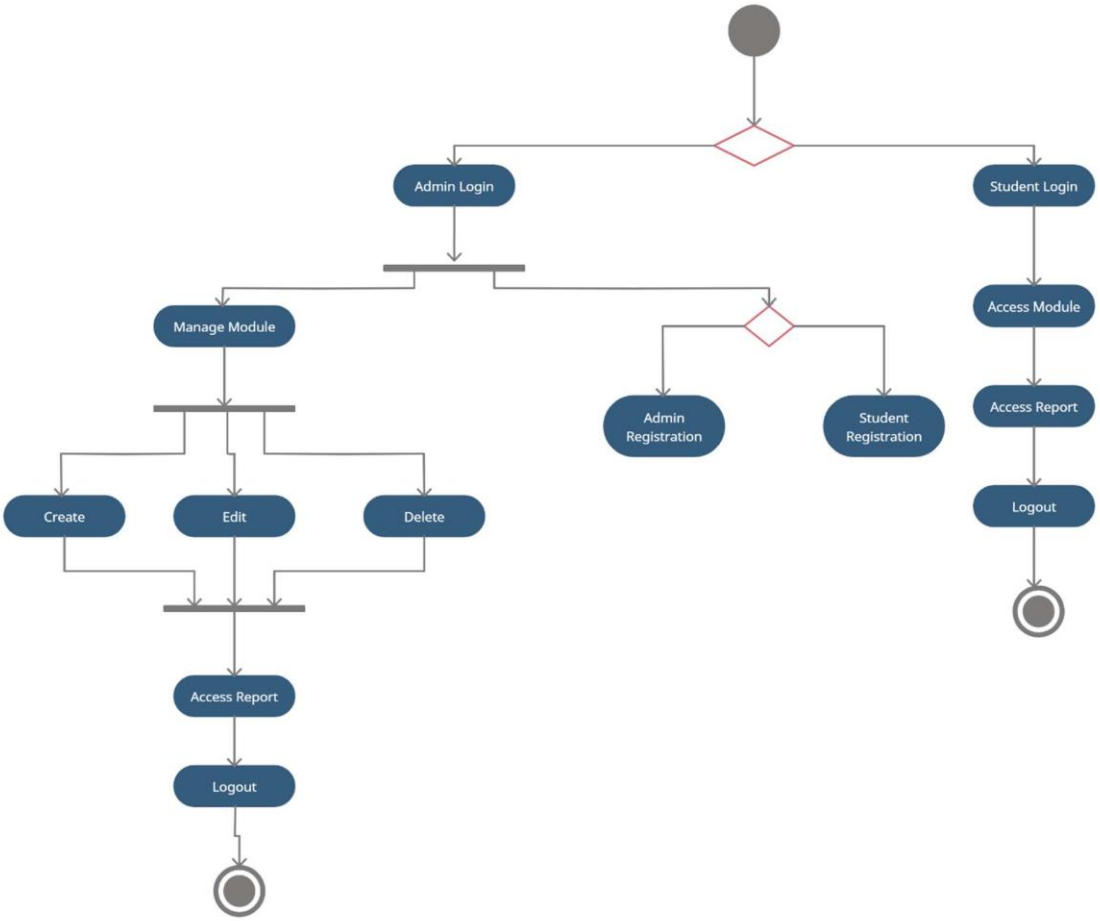


Figure – an activity diagram

The above diagram prints the number if it is odd otherwise it subtracts one from the number and displays it.

Activity Diagram:



Description:

The activity diagram encompasses two kinds of login systems: Student login and Admin login. The Student login is followed by permission to access the modules, access the reports and finally logout. The Admin login further consists of two more divisions: One that handles modules and another that handles the user creation. The Admin can register a Student or an Admin object if they choose to. As regards the module, they can manage the modules using three functionalities, namely create, edit and delete. They can also access the reports produced for every single Student that accessed the modules. That process is followed by the logout process.

UML OBJECT DIAGRAM

Definition:

An object diagram in the Unified Modelling Language (UML), is a diagram that shows a complete or partial view of the structure of a modelled system at a specific time.

Procedure:

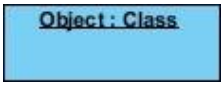
Object is an instance of a class in a particular moment in runtime that can have its own state and data values. Likewise, a static UML object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time, thus an object diagram encompasses objects and their relationships which may be considered a special case of a class diagram or a communication diagram.

The use of object diagrams is fairly limited, mainly to show examples of data structures.

- During the analysis phase of a project, you might create a class diagram to describe the structure of a system and then create a set of object diagrams as test cases to verify the accuracy and completeness of the class diagram.
- Before you create a class diagram, you might create an object diagram to discover facts about specific model elements and their links, or to illustrate specific examples of the classifiers that are required.

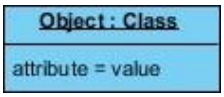
Object Names:

- Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon.



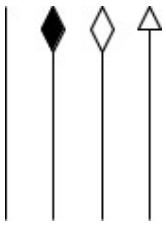
Object Attributes:

- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.

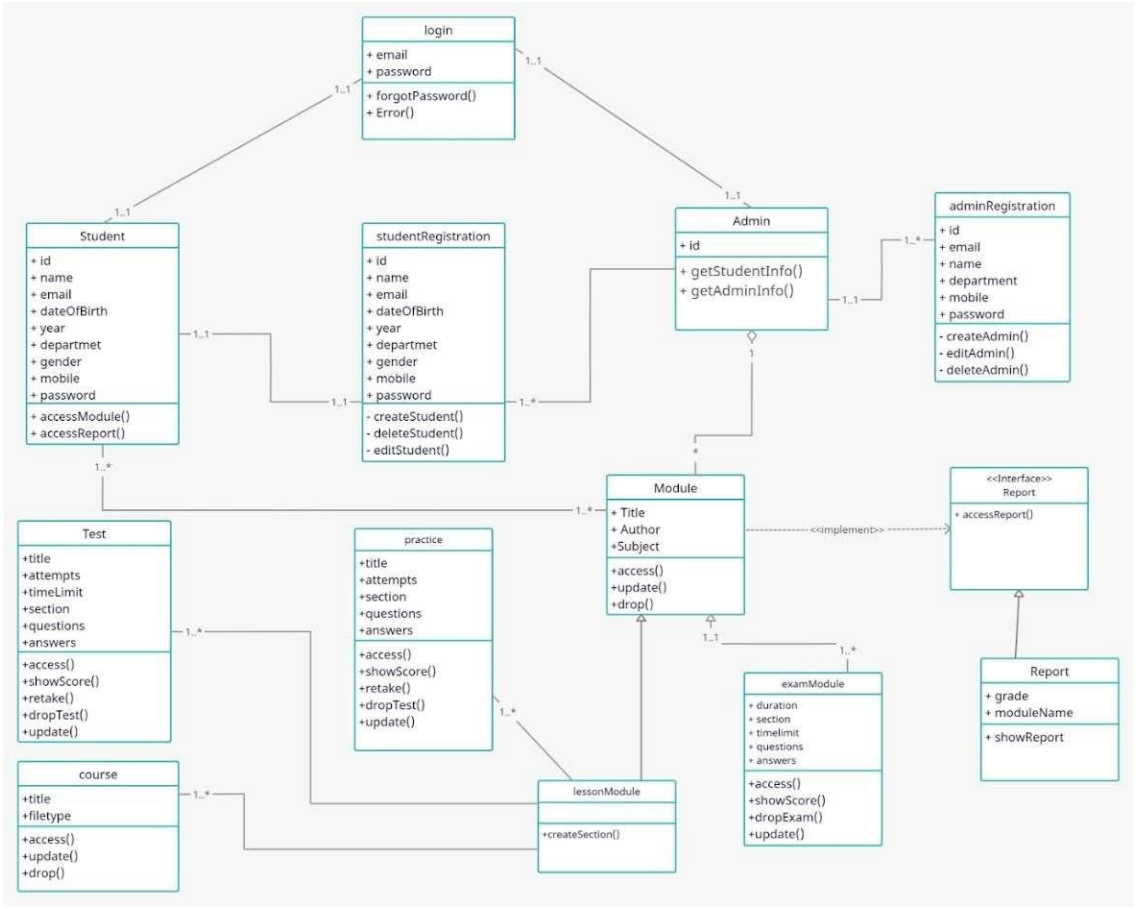


Links:

- Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.



Object Diagram:



Description:

The studentRegistration and adminRegistration classes are used to create objects Student and Admin respectively. The Admin object can create an object of the Module class and the Student object can access it. The Module object can be created by the Admin and can be accessed by the Student. Similarly, the Course and ExamModule objects can be created by the User. The

Student objects has the permission to only access the course material and attempt modules and practice tests. However, they can’t alter the course or module content.

UML STATE MACHINE DIAGRAM

Definition:

State-transition diagrams describe **all of the states** that an object can have, the events under which an object changes state (transitions), the conditions that must be fulfilled before the transition will occur (guards), and the activities undertaken during the life of an object (actions).

Procedure:

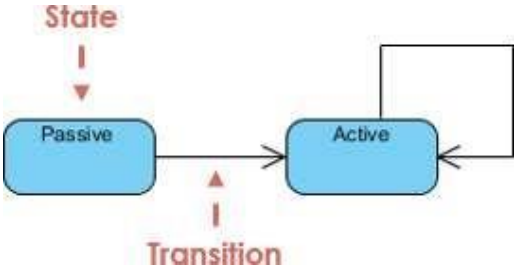
A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives. States refer to the different combinations of information that an object can hold, not how the object behaves. In order to understand the different 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.

There are several characteristics of states in general, regardless of their types:

- A state occupies an interval of time.
- A state is often associated with an abstraction of attribute values of an entity satisfying some condition(s).
- An entity changes its state not only as a direct consequence of the current input, but it is also dependent on some past history of its inputs.

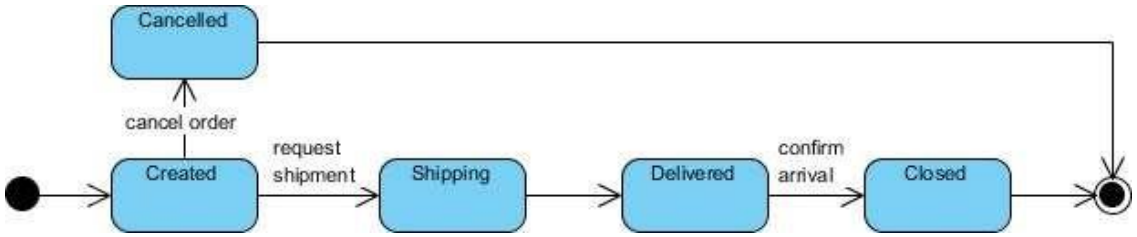
State

A state is a constraint or a situation in the life cycle of an object, in which a constraint holds, the object executes an activity or waits for an event.

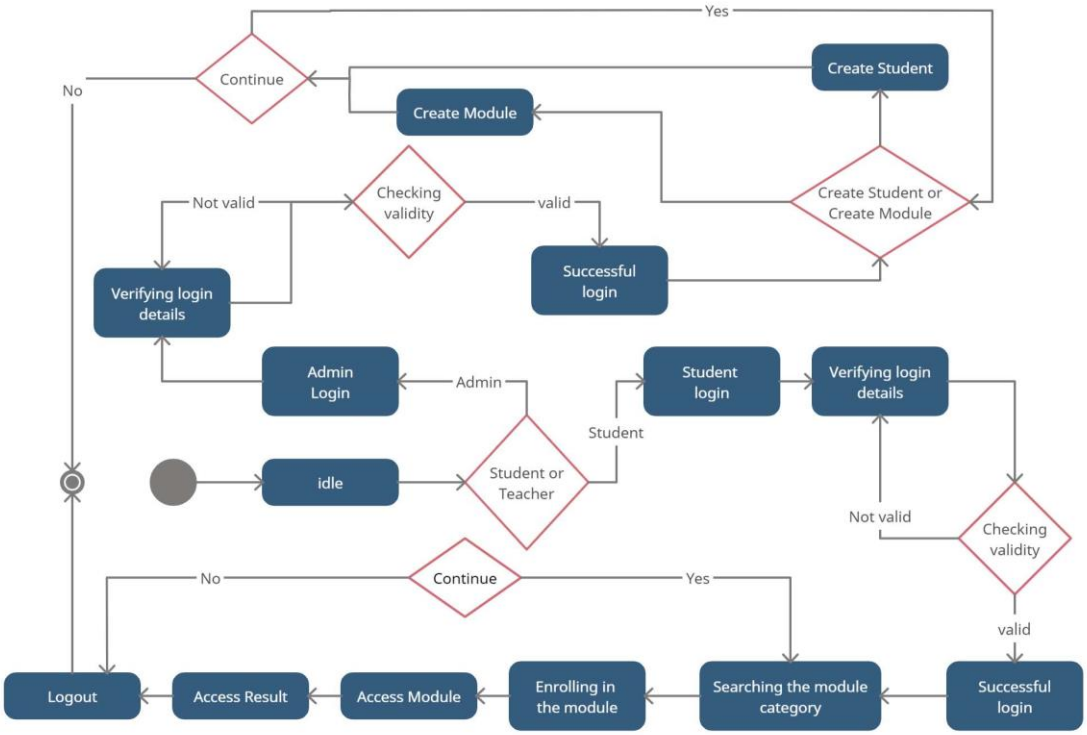


Initial and Final States

- The **initial state** of a state machine diagram, known as an initial pseudo-state, is indicated with a solid circle. A transition from this state will show the first real state
- The **final state** of a state machine diagram is shown as concentric circles. An open loop state machine represents an object that may terminate before the system terminates, while a closed loop state machine diagram does not have a final state; if it is the case, then the object lives until the entire system terminates.



State Transition Diagram:



Description:

The system's initial state is Start state. After the system starts it transitions into idle state until the user does login using either their Student or Admin account details. From that point it can take two routes based on the type of the account.

If it's a student account then the state transitions into Student login and eventually the state for verifying login details. The validity checking state follows the state which keeps looping back to the previous state until the condition is met. From there the Student can transition into searching state where the application can look for the course and that is followed by states enrol in module, access module and access result states. If the Students doesn't choose to continue then the state transitions to logout and then the end state.

If it's an admin account then the state transitions into admin login followed by the verifying login details. Once the condition is met the state transitions into successful login and then followed by a conditional state whether to create a student or create a module. The following

state is create student state or create module state depending on their response. If the Admin wishes to not to continue then the state transitions into end state.

UML DEPLOYMENT DIAGRAM

Definition:

A deployment diagram is a UML diagram type that **shows the execution architecture of a system**, including nodes such as hardware or software execution environments, and the middleware connecting them.

Procedure:

A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modelling the physical aspects of an object-oriented system. They are often be used to model the static deployment view of a system (topology of the hardware).

Deployment diagrams are important for visualizing, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

A deployment diagram is just a special kind of class diagram, which focuses on a system's nodes. Graphically, a deployment diagram is a collection of vertices and arcs. Deployment diagrams commonly contain:

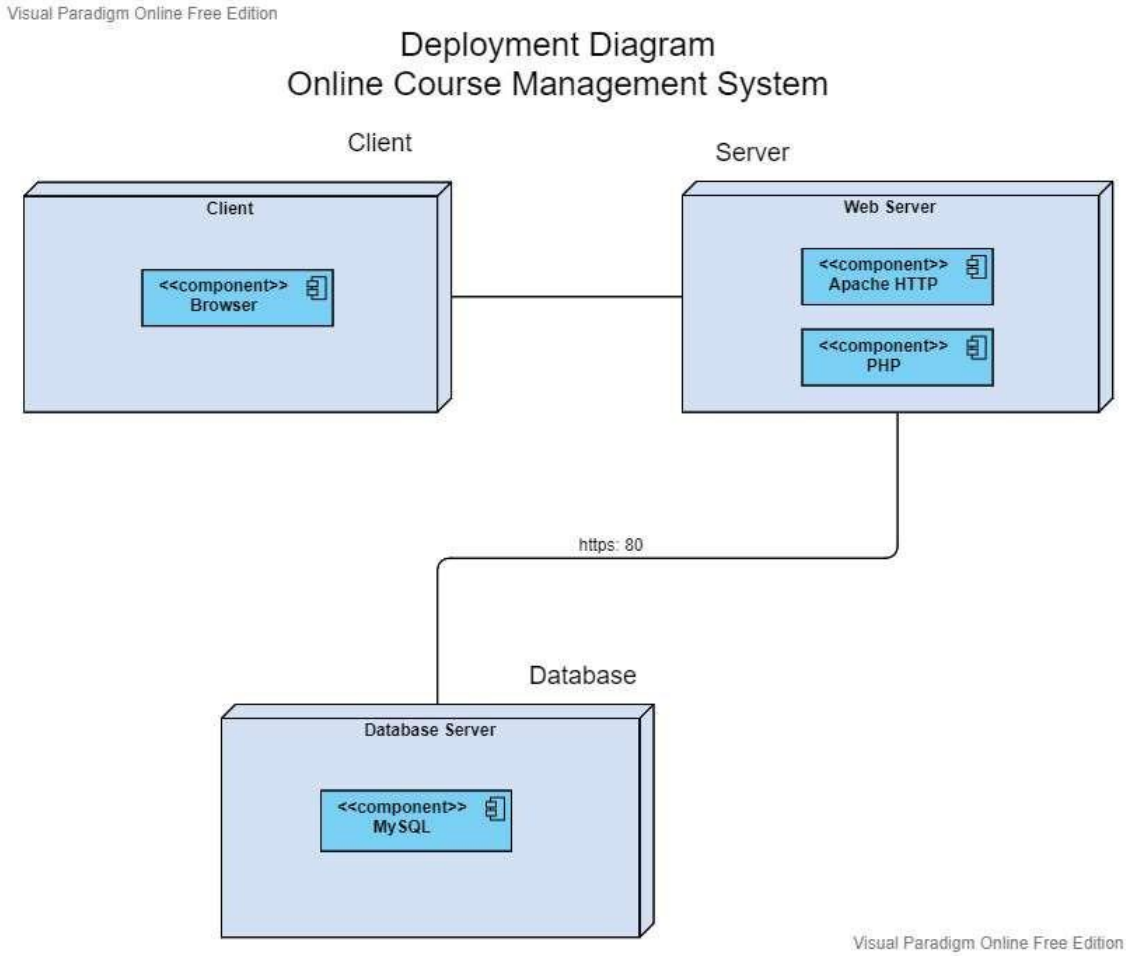
Nodes

- 3-D box represents a node, either software or hardware
- HW node can be signified with <<stereotype>>
- Connections between nodes are represented with a line, with optional <<stereotype>>
- Nodes can reside within a node

Other Notations

- Dependency
- Association relationships.
- May also contain notes and constraints.

Deployment Diagram:



Description:

The Deployment diagram of the Online Course Management System is split into three nodes: The Client node, the Web Server node, and the Database node. The Database node (Database server) is where we handle the database operations and deploy the database. The Web Server node handles the request-response intercommunication between the client and server using HTTP protocol. The server-side scripting is also included in this node. The Client node consists of the browser which is generally the client of the application that is accessing the server of the application.

UML COMPONENT DIAGRAM

Definition:

A component diagram, also known as a UML component diagram, **describes the organization and wiring of the physical components in a system**. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

Procedure:

A component diagram breaks down the actual system under development into various high levels of functionality. Each component is responsible for one clear aim within the entire system and only interacts with other essential elements on a need-to-know basis.

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. In UML 2, a component is drawn as a rectangle with optional compartments stacked vertically. A high-level, abstracted view of a component in UML 2 can be modeled as:

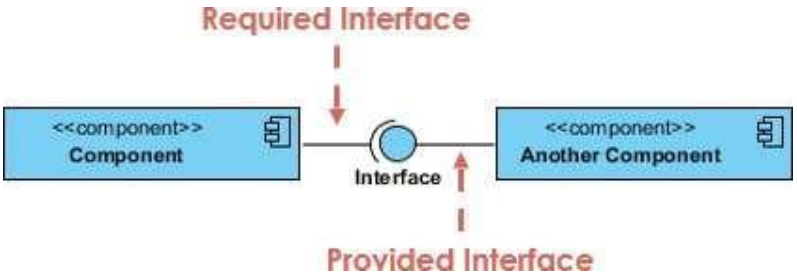
1. A rectangle with the component's name
2. A rectangle with the component icon
3. A rectangle with the stereotype text and/or icon

Interface

In the example below shows two type of component interfaces:

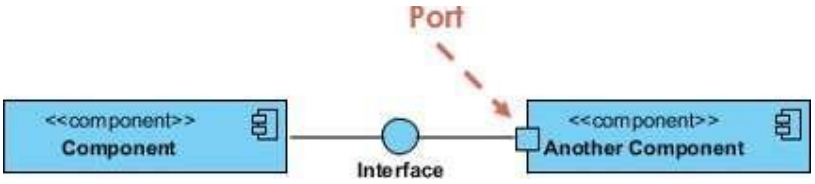
Provided interface symbols with a complete circle at their end represent an interface that the component provides - this "lollipop" symbol is shorthand for a realization relationship of an interface classifier.

Required Interface symbols with only a half circle at their end (a.k.a. sockets) represent an interface that the component requires (in both cases, the interface's name is placed near the interface symbol itself).

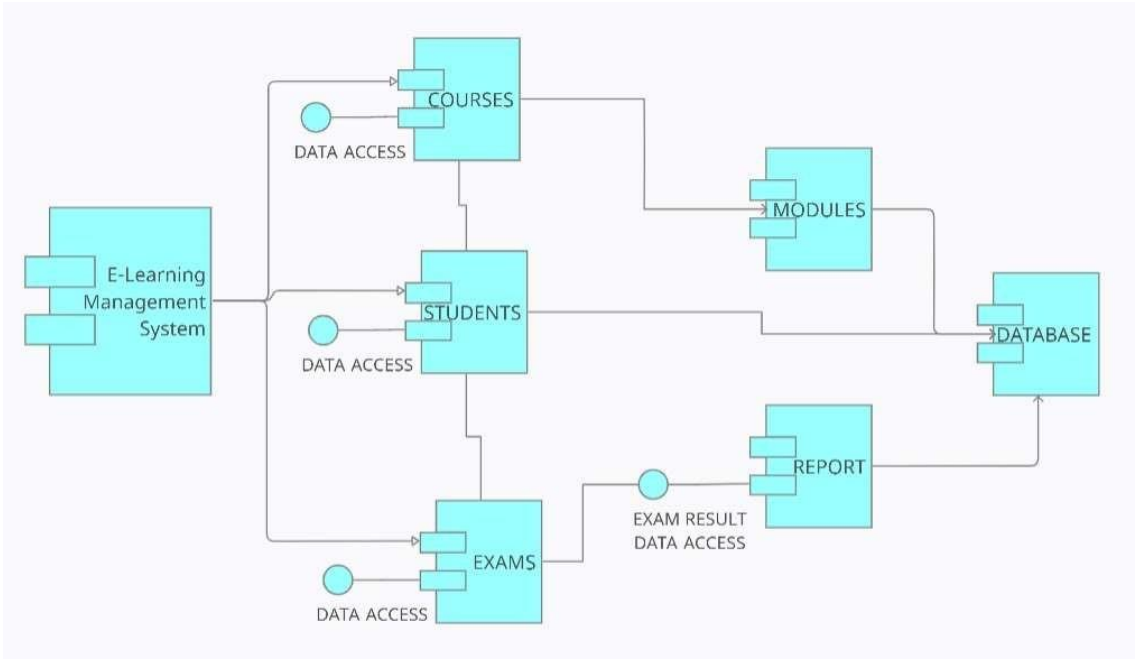


Port

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.



Component Diagram:



Description:

The main component of the system is the E-learning management system component. The main component branches out into three different components called Courses, Students and Exams all of which use the Data Access interface. The Data Access interface is responsible for enabling the components to access the information stored in the database. The Courses component is associated with the Modules component and the Exam component is associated with Report component that use the Exam Result Data Access interface. This interface allows the component to access the results generated by the Exams component. All the components finally converge into the Database component which supplies the other components with the data required for working.

UML PACKAGE DIAGRAM

Definition:

A package diagram in the Unified Modelling Language depicts the dependencies between the packages that make up a model.

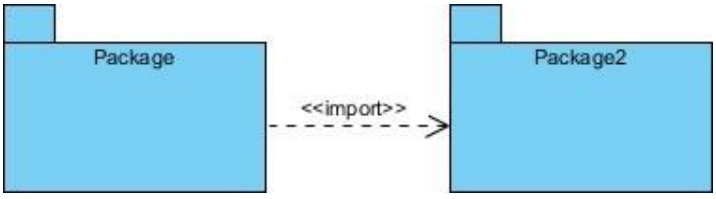
Procedure:

Package diagram, a kind of structural diagram, shows the arrangement and organization of model elements in middle to large scale project. Package diagram can show both structure and dependencies between sub-systems or modules, showing different views of a system.

There are two sub-types involved in dependency. They are <<import>> & <<access>>. Though there are two stereotypes' users can use their own stereotype to represent the type of dependency between two packages.

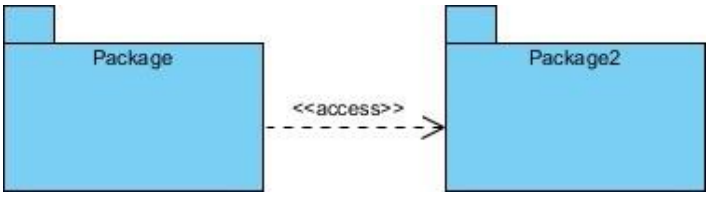
Package Diagram Example - Import

<<import>> - one package imports the functionality of other package

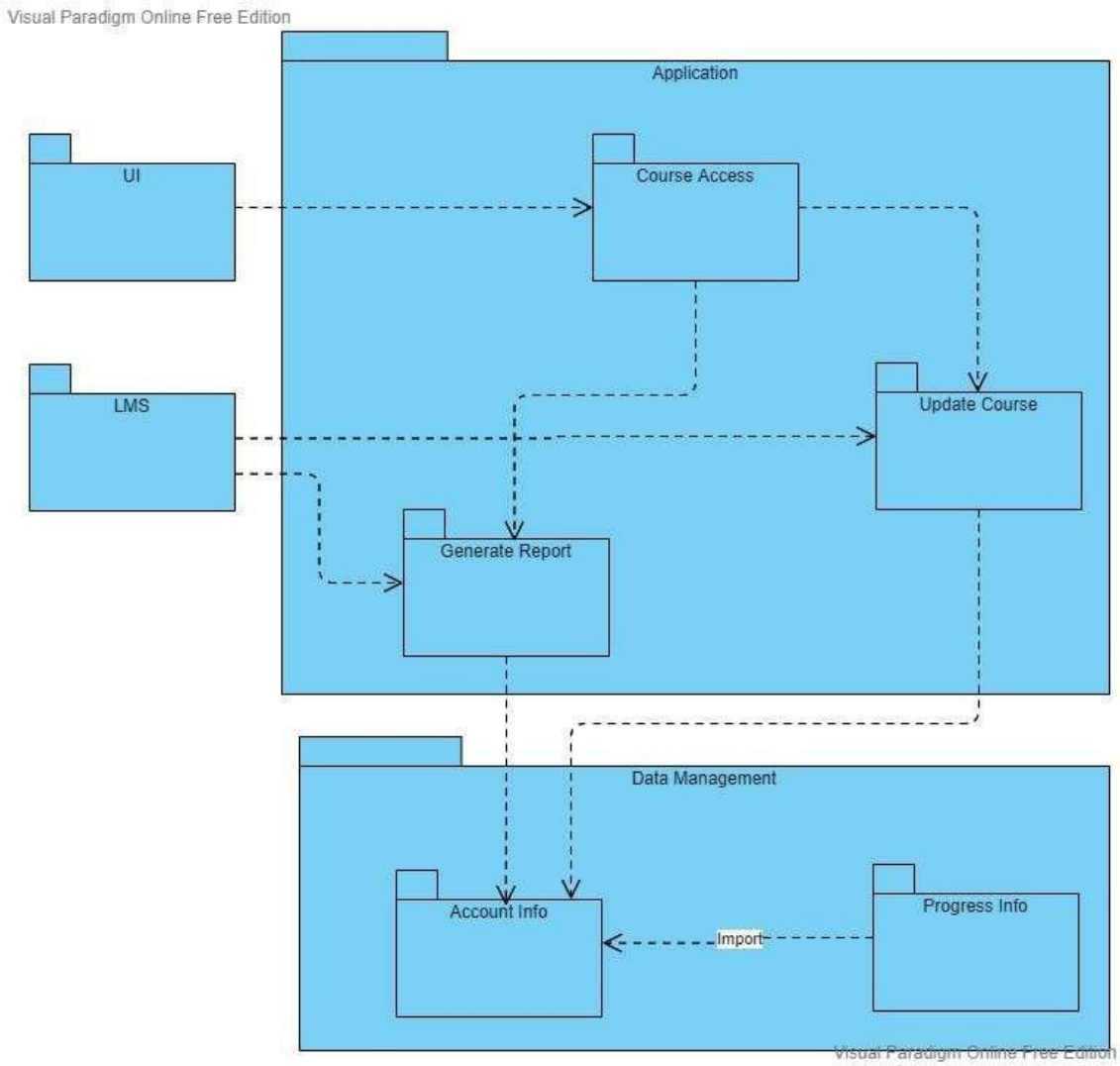


Package Diagram Example - Access

<<access>> - one package requires help from functions of other package.



Package Diagram:

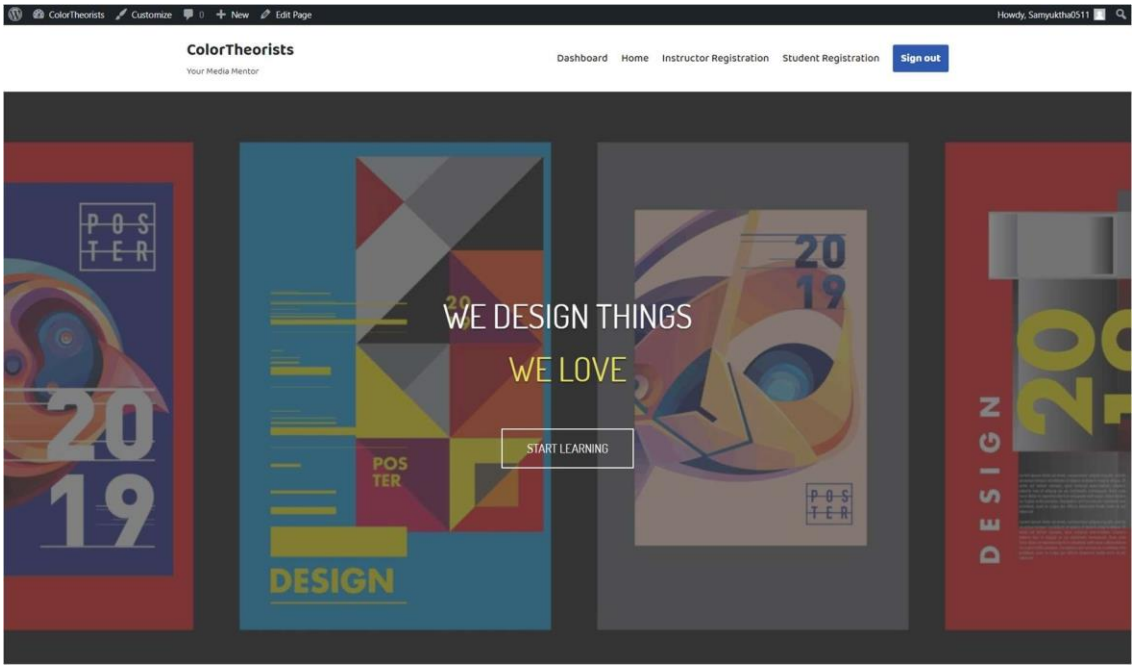


Description:

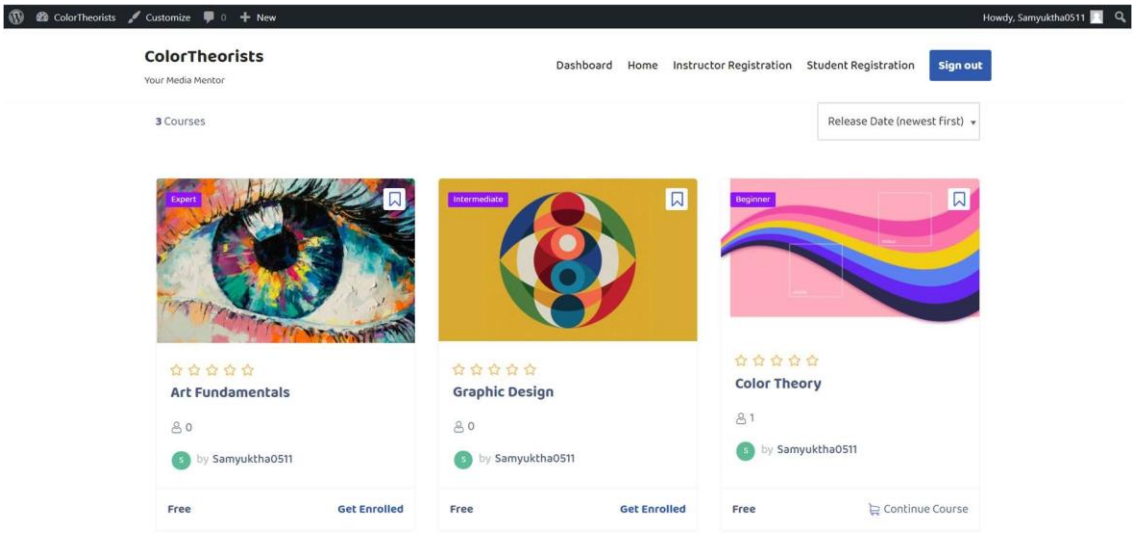
Package diagram of the online course management system has 4 top level packages: UI, LMS, Application and Data Management. The data management package encompasses the account information and progress information of the users. The progress information imports the account information package since the progress data is generally piggybacked on the user information.

The application package encompasses three packages: Course Access, Update Access and Generate Report. The UI package can interact with the Course Access package which can take course to interact with either the Generate Report (if the user is a Student) or Update Course package (if the user is an Admin). The latter packages are invoked using the LMS package that broadly represents the inner working of the application.

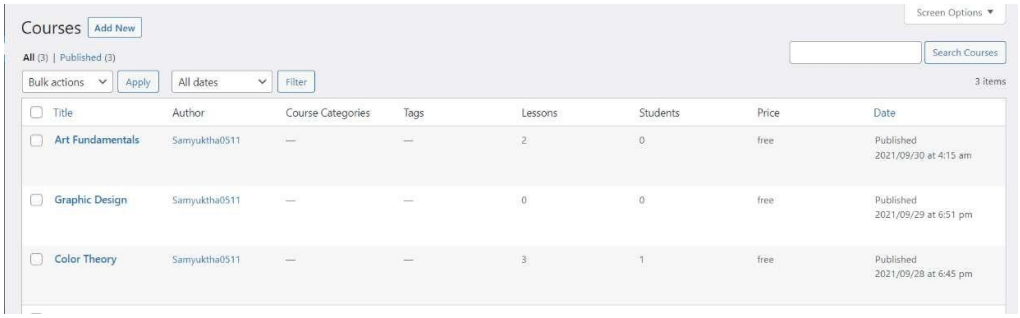
User Interfaces



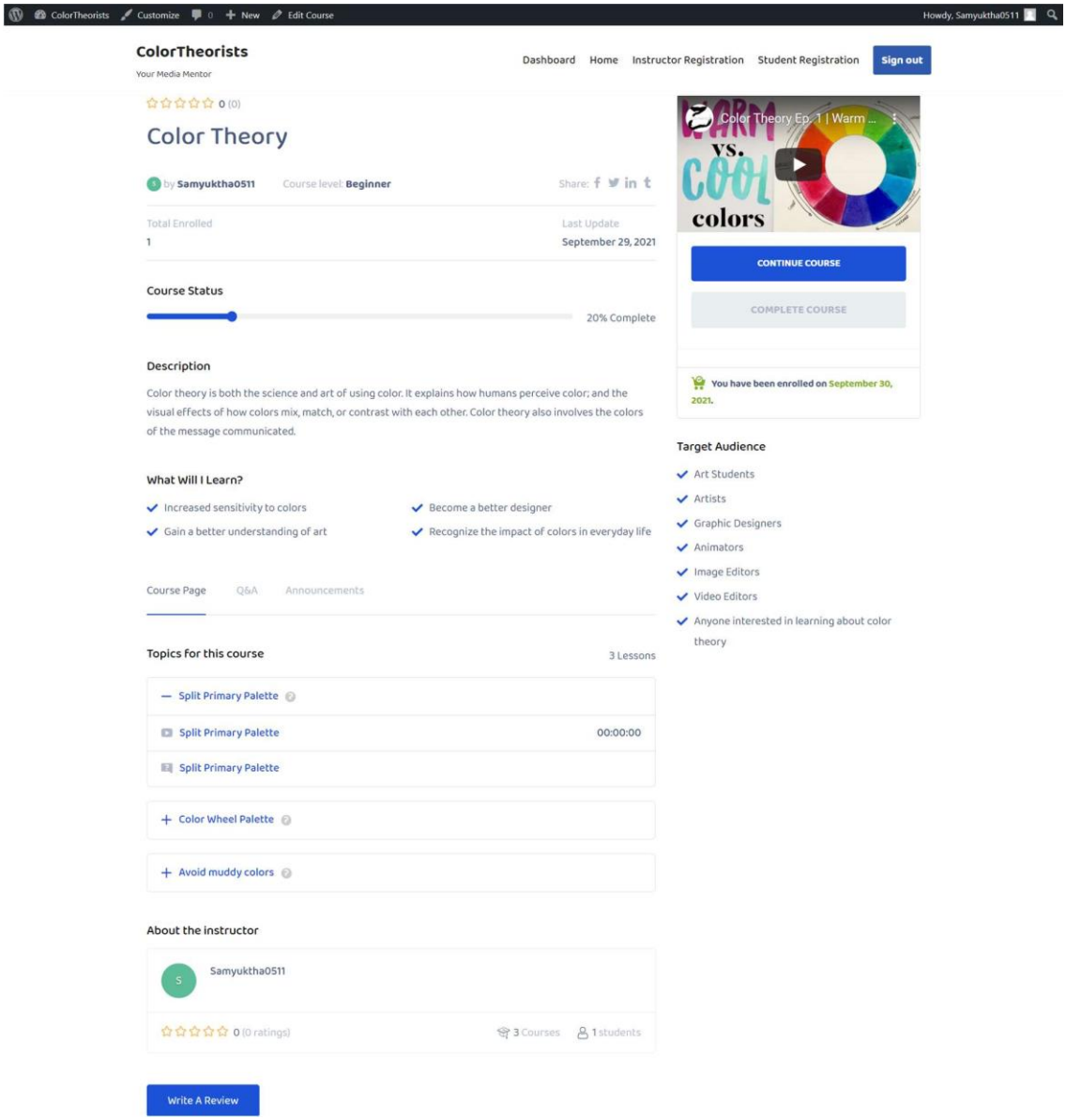
Home Page



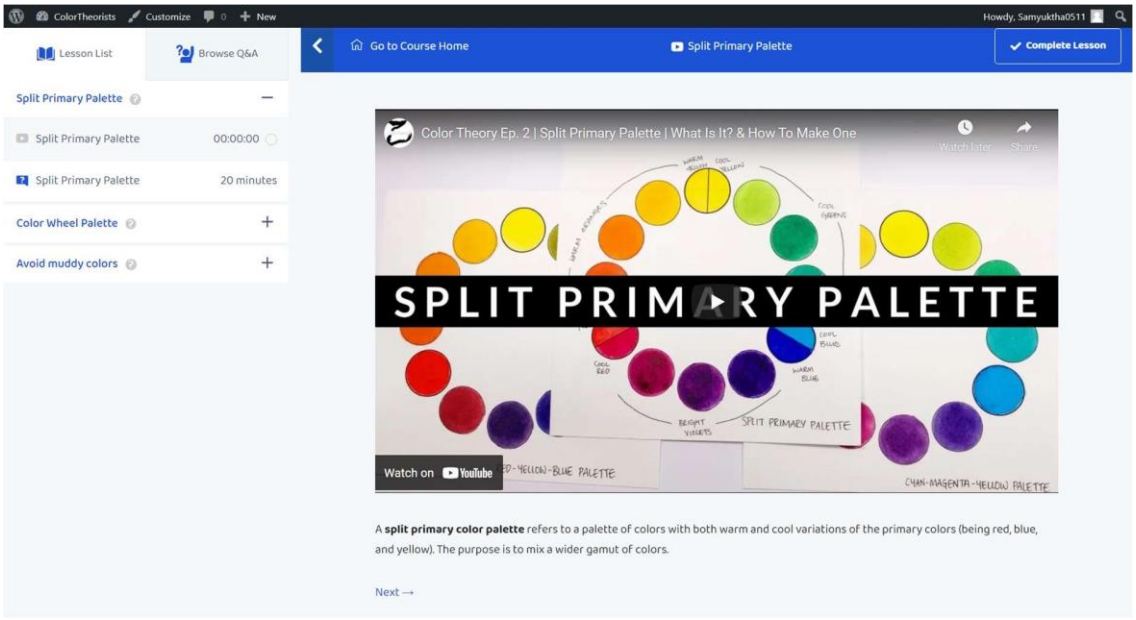
Courses: On clicking “the Start Learning” Button



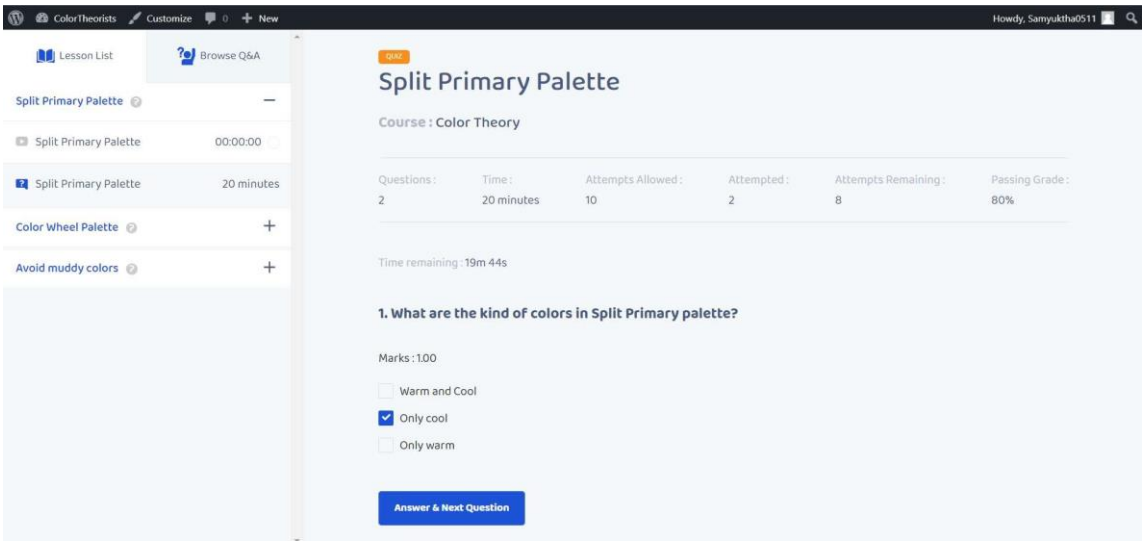
List of Courses



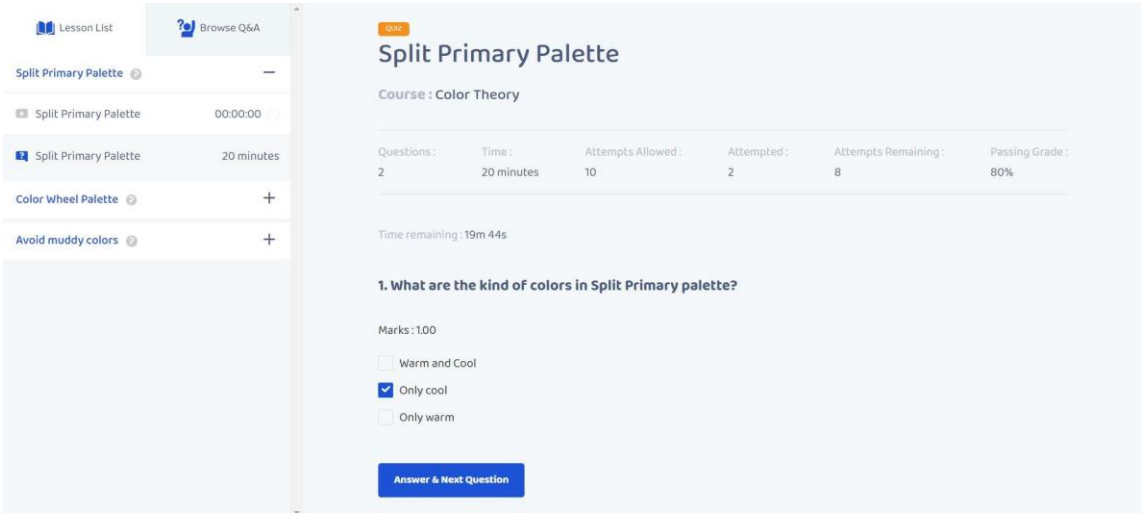
Course: Inside View



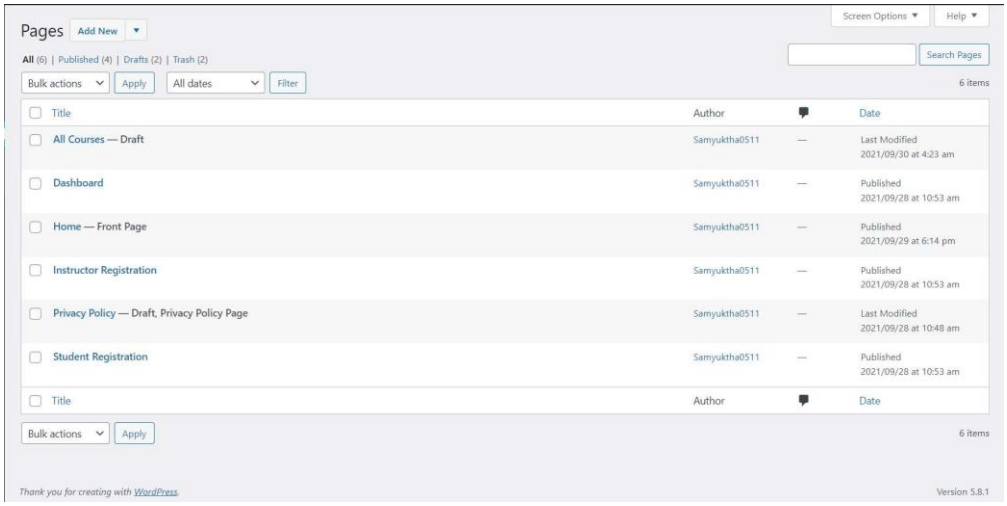
Course Modules



Course Quiz



Dashboard



Pages