Experiment 1: Apply the knowledge of SRS and prepare Software Requirement Specification (SRS) document in IEEE format for the project

Learning Objective: Students will be able to List various hardware and software requirements, distinguish between functional and nonfunctional requirements, indicate the order of priority for various requirements, analyze the requirements for feasibility.

Tools: IEEE template and MS Word

Theory:

The srs should contain the following Table of Contents

Introduction ....
Purpose ....
Intended Audience and Reading Suggestions................................
Product Scope ....
Reference
Overall Description...
Product Perspective.
Product Functions ....
User Classes and Characteristics...
Operating Environment...
Design and Implementation Constraints ....
Assumptions and Dependencies..
External Interface Requirements ....
User Interfaces ....
Hardware Interfaces ....
Software Interfaces...
Communications Interfaces...
System Features ....
System Feature 1 ....
Feature 2 (attach further requirements)..
Other Nonfunctional Requirements...
Performance Requirements...
Safety Requirements ....
Security Requirements.. .
Business Rules ....
Other Requirements ....
Appendix A: Glossary...
Prepared by,...

Software Engineering
Software Requirements Specification (SRS) Document

Shrenik Nair
Sandeep Maurya

10/02/2023

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft V.0	*Shrenik Nair* *Sandeep Maurya*	All sections being Filled	17/02/23

Review & Approval

Requirements Document Approval History

Approving Party	Version Approved	Signature	Date
Shrenik Nair			
Sandeep Maurya			

Requirements Document Review History

Reviewer	Version Reviewed	Signature	Date

Contents

1. Introduction

Introduction

The purpose of this document is to define and describe the requirements of the project and to spell out the system's functionality and its constraints.

Scope of this Document

This was a project made by students for educational purposes, but it is completely open source, so it is free to use for anyone that needs it.

Overview

The product is supposed to be an education website between kids and the website that we choose to connect to.

Business Context

This was a project made by students for educational purposes, but it is completely open source, so it is free to use for anyone that needs it, and had no funding for the entirety of the development process.

2. General Description

2.1 Product Functions

The product is supposed to take input from the Arduino devices and display them onto a well-made GUI that is easy to read for people without prior knowledge of coding.

2.2 Similar System Information

This program was developed using HTML5, CSS3, JS using UI elements that are very easy to render onto a display, hence take very little system requirements since vanilla was used.

2.3 User Characteristics

The users are kids who are interested in learning about monuments, when playing minigames to understand and learn about monuments about our country.

2.4 User Problem Statement

This was a project made by students for educational purposes, but it is completely open source, so it is free to use for anyone that needs it. This implies that there wasn't any actual user that is using our skills to develop this application.

2.5 User Objectives

The user would want a project that allows them to take values from the peripherals connected to the Arduino device and use that to gauge if there is any discrepancy or an emergency in any scenario that is thrown at it.

2.6 General Constraints

When developing a website, it's important to consider user experience, accessibility, security, performance, compatibility, scalability, and maintenance. These constraints ensure the website is effective, user-friendly, and can handle growth over time.

3. Functional Requirements

- User Experience: The website should be designed with the user in mind, making it easy to navigate, visually appealing, and interactive.

- Accessibility: The website should be accessible to all users, including those with disabilities. This includes features such as alternative text for images, keyboard navigation, and readable fonts.

- Security: The website should be secure, with measures in place to protect sensitive data such as user information and payment details.

- Performance: The website should be fast and responsive, with quick load times and smooth page transitions.

- Compatibility: The website should be compatible with different devices and browsers, ensuring that it can be accessed by as many users as possible.

- Scalability: The website should be designed to handle increased traffic and data as it grows over time.

- Maintenance: The website should be easy to maintain and update, with clear documentation and easy-to-use tools for making changes.

4. Interface Requirements

4.1 User Interfaces

4.1.1 GUI
The GUI was made in HTML5,CSS3, JS, without any framework also it is very light on resources.
4.1.2 CLI
There is no command line interface
4.1.3 API
Leaflet library was used for the product
4.1.4 Diagnostics or ROM
It is uploaded on github for further diagnosis

4.2 Hardware Interfaces

The user needs to have a Display between the GUI and the hardware.

4.3 Communications Interfaces

Communication interfaces are not required for this application

4.4 Software Interfaces

User Interface (UI): The UI is the graphical interface that allows users to interact with the website. It includes elements such as menus, buttons, forms, and navigation bars.

Application Programming Interface (API): An API allows different software applications to communicate with each other. It enables developers to create web-based applications that can interact with other systems or services.

Web Services: Web services are a type of API that allow software applications to communicate over the web. They use standardized protocols such as HTTP to transfer data between different systems.

5. Performance Requirements

Compatibility: The website should be compatible with different devices, browsers, and operating systems, ensuring that it can be accessed by as many users as possible.

User experience: The website should provide a positive user experience, with intuitive navigation and clear, easy-to-use interfaces.

6. Other non-functional attributes

.

6.1 Performance

This includes factors such as response time, availability, and scalability, which were mentioned earlier.

6.2 Security

This includes measures in place to protect user data, prevent unauthorized access, and ensure compliance with regulations such as GDPR or HIPAA.

6.3 Usability

This includes aspects such as ease of navigation, clarity of content, and accessibility for users with disabilities.

6.4 Reliability

This includes measures to ensure that the website operates consistently and with minimal errors or downtime.

6.5 Compatibility

This includes ensuring that the website is compatible with a variety of devices, browsers, and operating systems.

6.6 Maintainability

This includes making the website easy to maintain and update, with clear documentation and coding practices that facilitate troubleshooting and problem-solving.

Learning Outcomes: Students should have the ability to

LO1: List various hardware and software requirements
LO2: Distinguish between functional and nonfunctional requirements
LO3: Indicate the order of priority for various requirements
LO4: Analyze the requirements for feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare SRS document

Conclusion: We were able to prepare SRS document successfully

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 2: Draw DFD (upto 2 levels)

Learning Objective: Students will able to identify the data flows, processes, source and destination for the project, Analyze and design the DFD upto 2 levels

Tools: LucidChart

Theory:

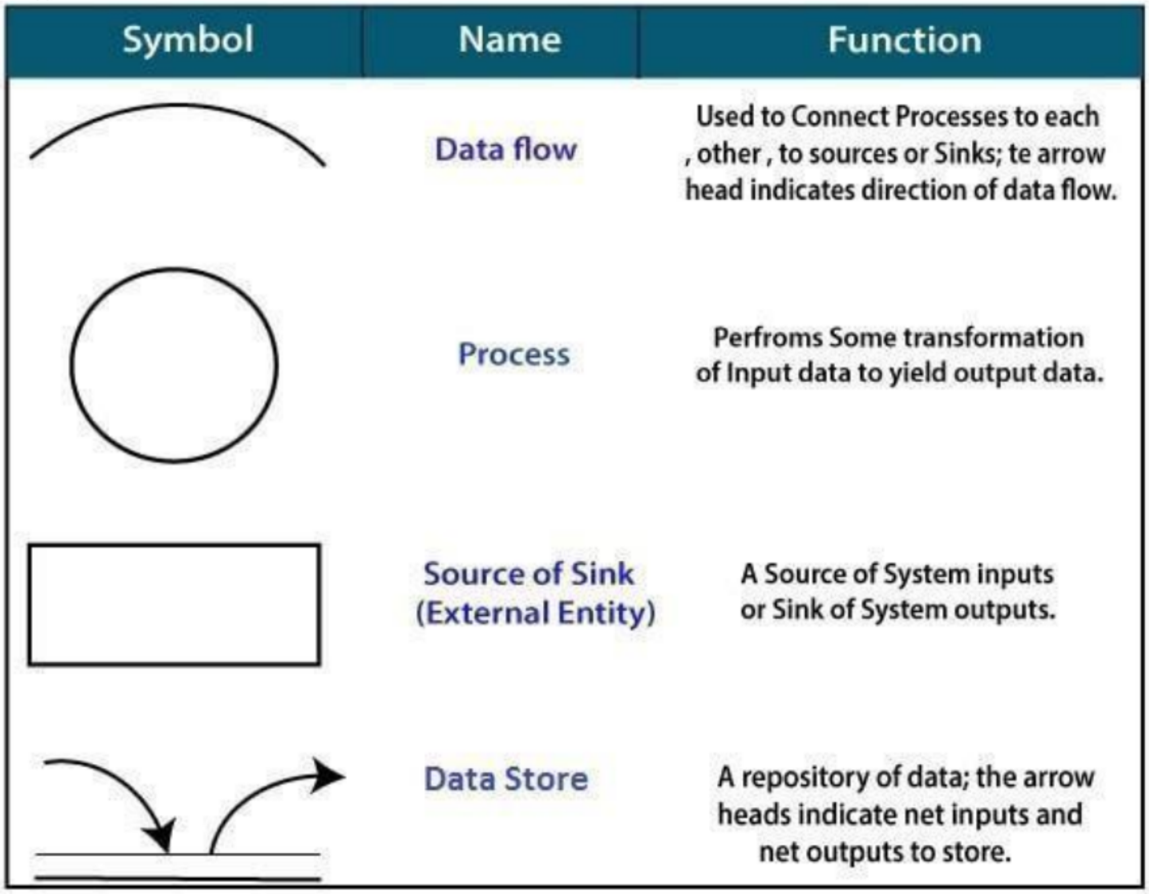
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.



Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Perfroms Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.
Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0- level DFDM

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming
and outgoing arrows. Then the system is decomposed and described as a DFD with multiple
Standard symbols for DFDs are derived from the electric circuit diagram analysis and are bubbles. Parts of the system represented by each of these bubbles are then decomposed
shown in fig:
and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x$_1$ and x$_2$ and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

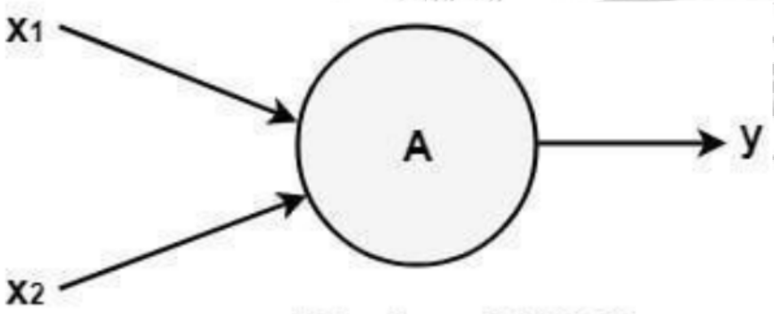


Fig: Level-0 DFD.

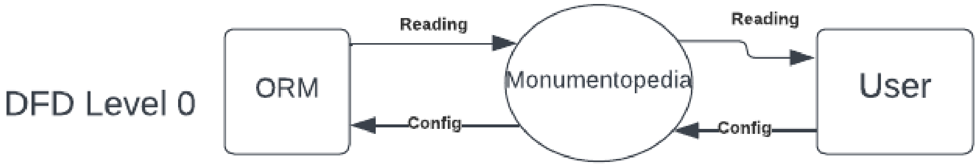
1- level DFD

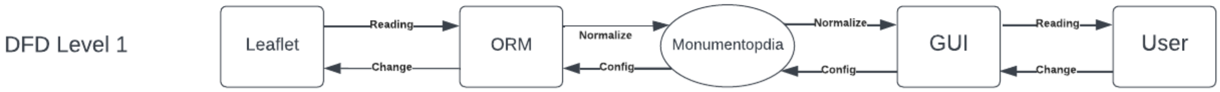
In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

2-Level DFD

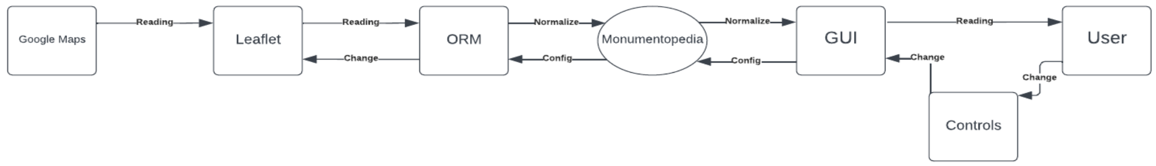
2- level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

OUTPUT:





DFD Level 2



Learning Outcomes: Students should have the ability to

LO1: Identify the dataflows, processes, source and destination for the project.

LO2: Analyze and design the DFD upto 2 levels

Outcomes: Upon completion of the course students will be able to prepare Draw DFD (upto 2 levels)

Conclusion: Successfully implemented dfd level 0, 1, 2 using lucidchart for project.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 03- Implement UML Use-case diagram

Learning Objective: To implement UML use-case diagram for the project

Tools: LucidChart

Theory:

Use case diagrams

Use case diagrams belong to the category of behavioral diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform one or more actors, and dependencies among them.

Actor

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

For example, consider the case where a customer *withdraws cash* from an ATM. Here, customer is a human actor.

Actors can be classified as below:

- **Primary actor**: They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
- **Supporting actor**: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the

diagram. **Use Case**

A use case is simply a functionality provided by a system.

Continuing with the example of the ATM, *withdraw cash* is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases include, *check balance*, *change PIN*, and so on.

Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

Subject

Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

Graphical Representation

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in figure - 01.

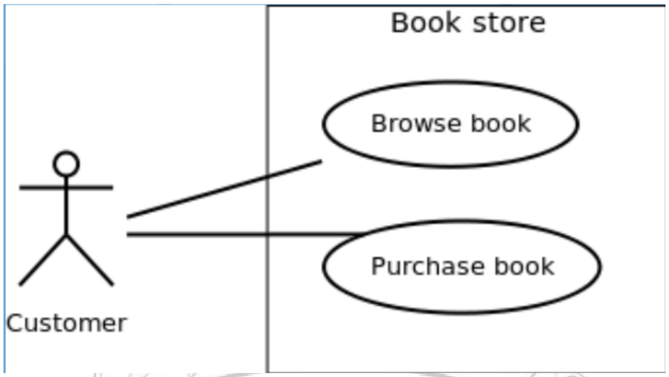


Figure - 01: A use case diagram for a book store

Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors is usually not shown. However, one can depict the class hierarchy among actors.

Use Case Relationships

Three types of relationships exist among use cases:

- Include relationship
- Extend relationship
- Use case generalization

Include Relationship

Include relationships are used to depict common behavior that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

Example

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a *login* use case, which is included by *compose mail*, *reply*, and *forward email* use cases. The relationship is shown in figure - 02.

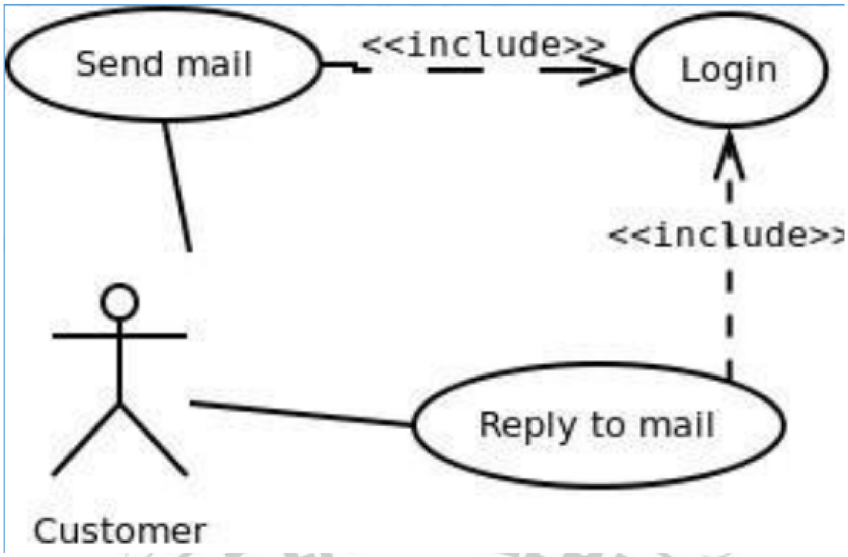


Figure - 02: Include relationship between use cases

Notation

Include relationship is depicted by a dashed arrow with a «include» stereotype from the including use case to the included use case.

Extend Relationship

Use case extensions are used to depict any variation to an existing use case. They are used to the specify the changes required when any assumption made by the existing use case becomes false.

Example

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows specifying any special shipping instructions, for example, call the customer before delivery. This *Shipping Instructions* step is optional, and not a part of the main *Place Order* use case. Figure - 03 depicts such relationship.

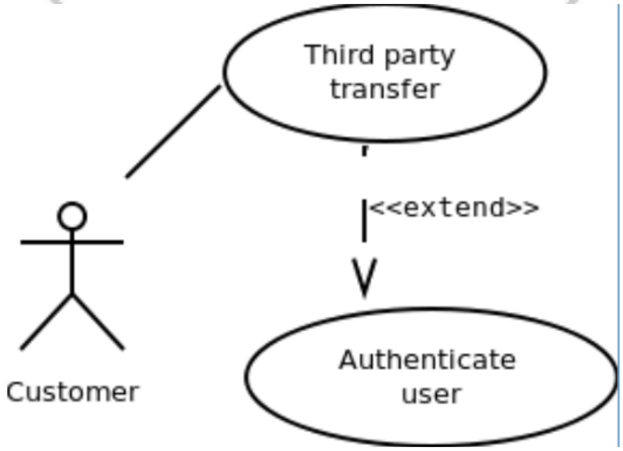


Figure - 03: Extend relationship between use cases

Notation

Extend relationship is depicted by a dashed arrow with a «extend» stereotype from the extending use case to the extended use case.

Generalization Relationship

Generalization relationship is used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case.

Example

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case *draw polygon*. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw polygon* and overrides its drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases. The relationship has been illustrated in figure - 04.

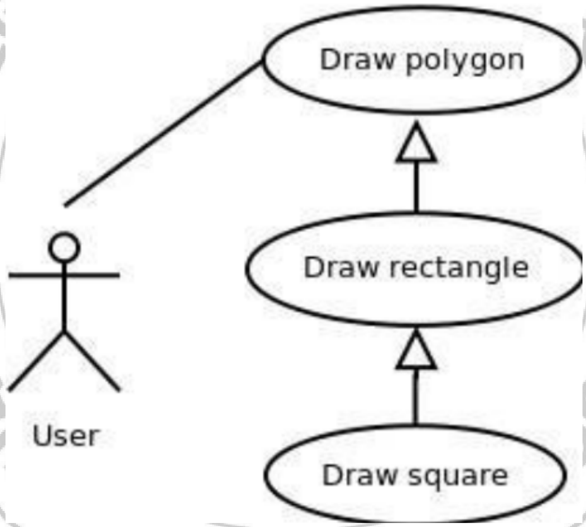


Figure - 04: Generalization relationship among use cases

Notation

Generalization relationship is depicted by a solid arrow from the specialized (derived) use case to the more generalized (base) use case.

Identifying Actors

Given a problem statement, the actors could be identified by asking the following questions :

- Who gets most of the benefits from the system? (The answer would lead to the identification of the primary actor)
- Who keeps the system working? (This will help to identify a list of potential users) •

What other software / hardware does the system interact with?

- Any interface (interaction) between the concerned system and any other system?

Identifying Use cases

Once the primary and secondary actors have been identified, we have to find out their goals i.e. what the functionality they can obtain from the system is. Any use case name should start with a verb like, "Check balance".

Guidelines for drawing Use Case diagrams
Following general guidelines could be kept in mind while trying to draw a use case diagram :

- Determine the system boundary
- Ensure that individual actors have well-defined purpose
- Use cases identified should let some meaningful work done by the actors
- Associate the actors and use cases -- there shouldn't be any actor or use case floating without any connection
- Use include relationship to encapsulate common behavior among use cases , if any

Procedure:

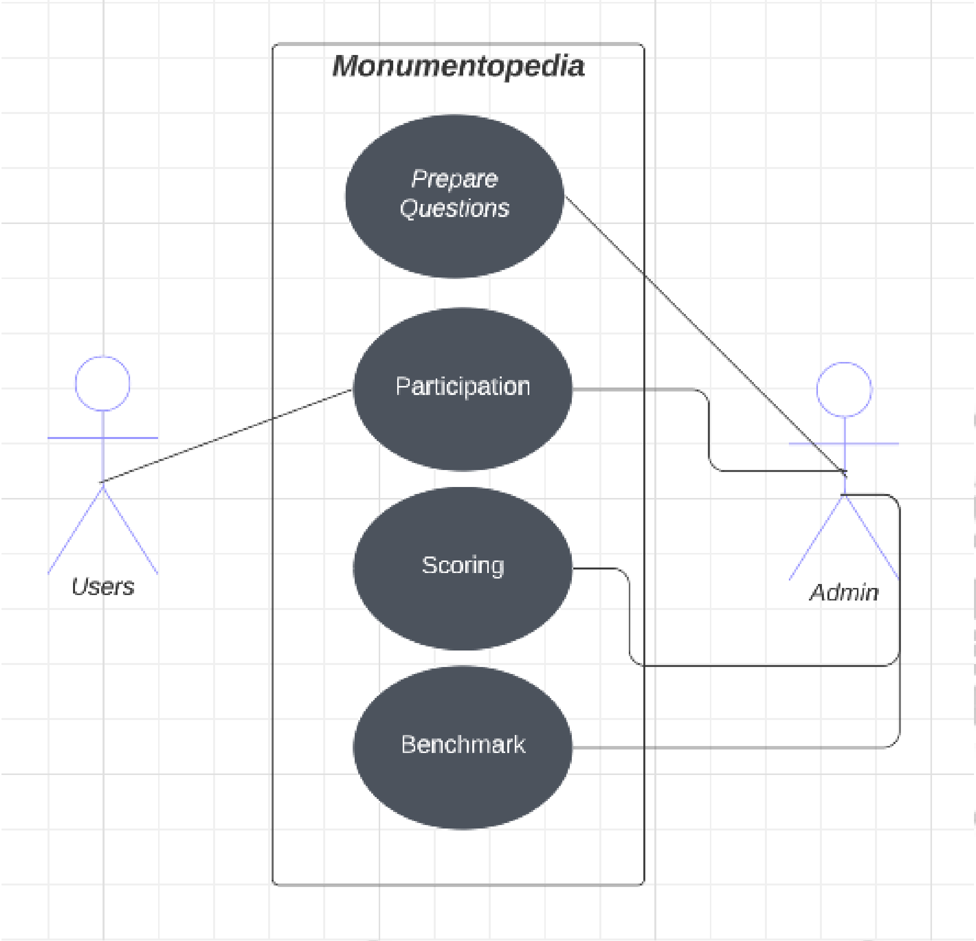
A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.

Result and Discussion:

Q.1) What is a use-case diagram? Draw at least two use-cases for your projects.

A use case diagram is a type of UML (Unified Modeling Language) diagram that depicts the interactions between a system and its users or other external systems. It shows the various use cases or ways in which the system is used by different actors, such as end-users, administrators, and external systems.



Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of use-case diagrams.

LO 2: Draw use-case diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate use-case diagrams.

Conclusion: Thus, students have understood and successfully drawn use-case diagrams.

Viva Questions:

1. What use-case are diagrams used for?
2. Enumerate the type of relationships that exist for use-case diagrams.

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				