International Institute of Professional Studies

DAVV-INDORE

Lab-Assignment

Roll Number:IT-2K19-53

Class: M.TECH.(I.T.)-2K19

Subject: Object Oriented Analysis and Design

Submitted to: Submitted by:

Sonali K. Mam Sanidhya Gupta

# OBJECT-ORIENTED ANALYSIS AND DESIGN

**LAB FILE**

# Que 1. What is OOAD?

OOAD stands for Object-Oriented Analysis and Design. It is a methodology that uses object-oriented concepts for analyzing and designing a system. Object-oriented programming (OOP) is a programming paradigm that relies on the concept of "objects," which can encapsulate data and behavior.

In the context of OOAD, the process involves breaking down a system into a collection of interacting objects that collaborate to achieve a set of goals. The key principles of OOAD include:

* Objects: Everything in the system is considered as an object, which is an

instance of a class. Objects have both data (attributes or properties) and behavior (methods or functions).

* Classes: A class is a blueprint for creating objects. It defines a set of attributes and methods that the objects instantiated from the class will have.
* Abstraction: Abstraction involves simplifying complex systems by modeling classes based on the essential properties and behaviors they share.
* Encapsulation: Encapsulation is the bundling of data and the methods that operate on that data into a single unit or class. It helps in hiding the internal details of an object and exposing only what is necessary.
* Inheritance: Inheritance allows a new class (subclass or derived class) to inherit attributes and methods from an existing class (base class or parent class). It promotes code reusability and the creation of a hierarchy of classes.
* Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables the use of a single interface to represent different types of objects.

# Que 2. Best practices of OOAD?

Object-Oriented Analysis and Design (OOAD) involves various best practices to ensure the development of robust, maintainable, and scalable software systems. Here are some best practices in OOAD:

* Understand the Problem Domain:
* Before diving into design, thoroughly understand the problem domain. This includes understanding the requirements, constraints, and the context in which the system will operate.
* Modularity and Encapsulation:
* Encapsulate the internal details of an object and provide a clear and well-defined interface. This promotes modularity, making it easier to understand, maintain, and modify individual components without affecting the entire system.
* Abstraction:
* Use abstraction to model essential aspects of a system while hiding unnecessary details. This simplifies the system and makes it easier to comprehend and work with.
* Hierarchy and Inheritance:
* Use inheritance judiciously to promote code reuse and establish a hierarchy of classes. However, be mindful of the "is-a" relationship, ensuring that subclass objects are true extensions of their parent class.
* Polymorphism:
* Leverage polymorphism to allow objects of different classes to be treated uniformly through a common interface. This enhances flexibility and extensibility.
* Use Case and Scenario Modeling:
* Identify and model use cases to understand the interactions between actors and the system. Develop scenarios to illustrate how the system behaves in different situations.
* Consistent Naming Conventions:
* Adopt a consistent naming convention for classes, methods, variables, and other elements. This promotes readability and maintainability across the codebase.
* Dynamic Modeling:
* Utilize dynamic modeling techniques, such as sequence diagrams, state diagrams, and activity diagrams, to illustrate how objects interact over time. This provides a clearer understanding of the system's behavior.
* Design Patterns:
* Familiarize yourself with and apply design patterns where appropriate. Design patterns are proven solutions to recurring design problems and can improve the efficiency and maintainability of your code.
* Collaboration and Communication:
* Foster collaboration and effective communication among team members. OOAD is often a collaborative effort, and clear communication ensures that everyone has a shared understanding of the design and its goals.

# Que 3.What are UML diagrams? What is the primary goal of UML diagrams & types of diagrams with description.

UML diagrams are graphical representations of a system's structure and behavior and are used to communicate and document various aspects of a software application.

There are several types of UML diagrams, each serving a specific purpose. Some common types include:

* Class Diagrams: Depict the static structure of a system, showing classes, their attributes, and the relationships between classes.
* Use Case Diagrams: Illustrate the interactions between the system and its external actors (users or other systems) and the various use cases in which the system is involved.
* Sequence Diagrams: Show the interactions between objects over time, emphasizing the sequence of messages exchanged.
* Activity Diagrams: Represent workflows or processes, showing the flow of activities and actions within a system.
* State Machine Diagrams: Describe the different states a system or object can be in and how it transitions between those states.
* Component Diagrams: Display the high-level components of a system and the relationships between them.
* Deployment Diagrams: Illustrate the physical deployment of software components in a system, showing how they are distributed across hardware nodes.
* Object Diagrams: Provide a snapshot of a system at a particular point in time, focusing on specific instances of classes and their relationships.

The primary goal of UML diagrams is to provide a standardized and visual way to model and document the various aspects of a software system. UML, or Unified Modeling Language, serves as a common language for developers, analysts, designers, and other stakeholders involved in software development. The key objectives of UML diagrams include:

* Communication: UML diagrams facilitate communication among team members and stakeholders by providing a visual representation of the system's structure and behavior. They serve as a common ground for discussing and understanding complex software concepts, making it easier for people with different roles and backgrounds to collaborate effectively.
* Visualization: UML diagrams offer a graphical representation of software components, relationships, and processes. Visualization helps in understanding the system's architecture, design, and functionality, making it easier to identify patterns, potential issues, and improvement opportunities.
* Specification: UML diagrams provide a means to specify and document the requirements, design, and architecture of a software system. They help capture and communicate the essential characteristics and constraints of a system, enabling a shared understanding among project stakeholders.
* Analysis and Design: UML supports the analysis and design phases of software development. Different types of diagrams within UML, such as class diagrams, use case

diagrams, and sequence diagrams, help in modeling the static and dynamic aspects of a system, allowing developers to make informed design decisions.

* Documentation: UML diagrams serve as a form of documentation for software systems. They offer a visual representation that complements textual documentation, making it more comprehensive and accessible. This documentation is valuable for maintaining and evolving the software over time.
* Standardization: UML provides a standardized set of notations and semantics for modeling software systems. This standardization helps ensure consistency and clarity in communication across different projects and organizations. It also facilitates the exchange of models and information between tools that support UML.

Types of diagrams -

Here's an overview of some common types of UML diagrams, along with brief descriptions of each:

Class Diagram:

* + Description: Illustrates the static structure of a system by showing classes, their attributes, methods, and the relationships between classes.
  + Use: Used for conceptual modeling, understanding the system's architecture, and identifying class hierarchies.

Use Case Diagram:

* + Description: Represents the interactions between a system and its external actors (users or other systems), showcasing various use cases and their relationships.
  + Use: Useful for capturing and communicating high-level system functionalities from a user's perspective.

Sequence Diagram:

* + Description: Shows the interactions between different objects or components over time, emphasizing the sequence of messages exchanged.
  + Use: Valuable for understanding the dynamic behavior of a system, especially in scenarios involving multiple components.

Activity Diagram:

* + Description: Depicts the workflow or business process by representing activities, decisions, and transitions between different states.
  + Use: Useful for modeling business processes, system workflows, and complex algorithms.

Component Diagram:

* + Description: Illustrates the high-level components of a system and the relationships between them, emphasizing the physical or logical structure.
  + Use: Useful for understanding the system's architecture and the organization of its components.

Deployment Diagram:

* + Description: Represents the physical deployment of software components on hardware nodes, showing the relationships between software and hardware.
  + Use: Useful for understanding the distribution of components in a networked environment.

Interaction Diagram :

* + Interaction diagrams can be used to illustrate how components like the frontend, backend, database, and external services collaborate during scenarios such as algorithm exploration, chat interactions, and data updates.

# Que 4. What is problem statement ?

A problem statement is a concise and clear description of an issue or challenge that needs to be addressed. It is a fundamental part of problem-solving and project management, providing a detailed understanding of the problem to guide the development of solutions. A well-crafted problem statement typically includes the following elements:

* Context: Describes the background or context in which the problem exists. This helps to provide a clear understanding of the environment or situation surrounding the issue.
* Problem Description: Clearly defines the problem, its scope, and its impact. This section outlines what is not working or what needs improvement.
* Objective: States the goals or objectives of solving the problem. It explains what you hope to achieve by addressing the issue.
* Relevance: Explains why the problem is important and why it requires attention. This helps to justify the allocation of resources and effort to solve the problem.
* Constraints: Identifies any limitations or restrictions that may impact the solution. This includes factors such as time constraints, budget limitations, or technological constraints.
* Scope: Defines the boundaries of the problem, indicating what is included and excluded from consideration. This helps to focus efforts on the most critical aspects of the issue.
* Benefits of Solving the Problem: Highlights the potential positive outcomes or benefits that will result from solving the problem. This can include improvements in efficiency, cost savings, or enhanced performance.
* Audience: Specifies the target audience or stakeholders who are affected by the problem or who will benefit from its solution.

# Que 5. What is SRS ?

SRS stands for Software Requirements Specification. It is a comprehensive document that outlines the functional and non-functional requirements of a software system. The SRS serves as a communication bridge between the client or customer and the development team, providing a detailed understanding of what the software is expected to accomplish.

Here are key components typically found in a Software Requirements Specification:

* Introduction:
* Provides an overview of the document, its purpose, and the scope of the software system.
* Purpose:
* Clearly states the objectives and goals of the software project.
* Scope:
* Defines the boundaries of the software system, detailing what is included and what is excluded.
* Definitions, Acronyms, and Abbreviations:
* Provides a list of specialized terms and their meanings to ensure a common understanding among all stakeholders.
* References:
* Lists any external documents or references that are relevant to the software requirements.
* Overall Description:
* Presents a high-level view of the software system, including its functionality, user characteristics, constraints, and dependencies.
* Specific Requirements:
* Details the functional and non-functional requirements of the software. This section is often organized into sub - sections like:
* Functional Requirements: Describes the specific features and functions the software must perform.
* Non-Functional Requirements: Specifies criteria that are not related to specific behaviors, such as performance, security, and usability.
* External Interface Requirements:
* Describes how the software interacts with external entities, including users, hardware, software, and other systems.
* System Features:
* Provides a detailed breakdown of the major features and functionalities of the software system.
* Other Requirements:
* Outlines any additional requirements, such as performance requirements, design constraints, and quality standards.
* Appendix:
* Includes supplementary information, such as mock-ups, diagrams, or prototypes, to enhance the understanding of the requirements.

The SRS serves as a critical document throughout the software development life cycle. It guides the development team in creating a product that meets the client's expectations, and it provides a baseline for testing and validation activities. Regular updates to the SRS may be necessary as the project progresses and requirements evolve.

# Que 6. Diagrams with description.

1. Use- Case Diagram -

Creating a use case diagram involves identifying actors, use cases, and their relationships. In the context of a QR code generator, let's identify the actors and use cases based on the provided HTML pages.

Actors:

User: The person interacting with the QR code generator.

Use Cases:

Generate QR Code:

Actor: User

Description: The user enters data, selects options (color, background, heading), and clicks the "Generate" button to create a QR code.

Associations:

Uses the /generate endpoint on the server.

View QR Code Result:

Actor: User

Description: After generating the QR code, the user is presented with a result page displaying the QR code image and additional options.

Associations:

Utilizes the result.html page.

Download QR Code:

Actor: User

Description: The user selects the file type (PNG or PDF), enters optional heading and heading color, and clicks "Download" to save the QR code.

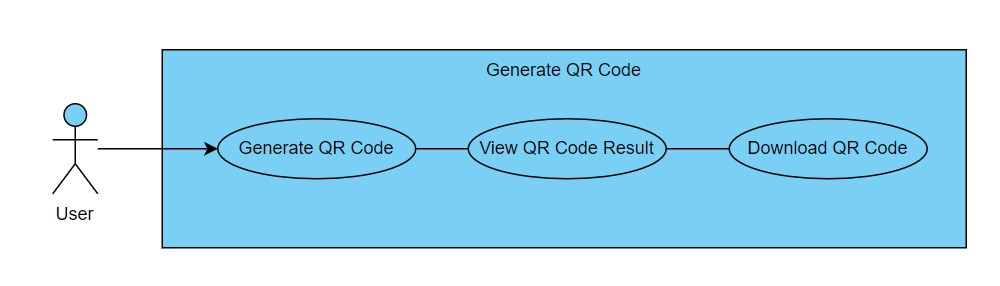


Fig: Use- Case Diagram for Project

1. Activity Diagram for Login Activity -

An activity diagram illustrates the flow of activities and actions within a system or process.

This simplified activity diagram outlines the basic flow of activities within the

"QR site" application, starting from the user selecting an action to the system processing user input and providing feedback, and finally ending the process. In a real-world scenario, the activities would be more detailed and include specific actions

and interactions. The diagram can be extended and refined to capture the full range of activities and interactions within your application.

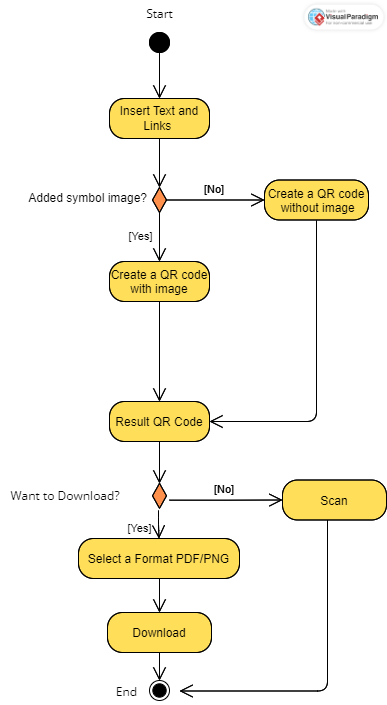


Fig: Activity Diagram for Project

1. Sequence Diagram-

A sequence diagram in the Unified Modeling Language (UML) represents the

interactions between objects or components in a system over time. It illustrates the order of messages exchanged between different entities and the flow of control in a particular scenario.

description of a sequence diagram for a user exploring an algorithm in the "QRSite-A QR code generator website”:

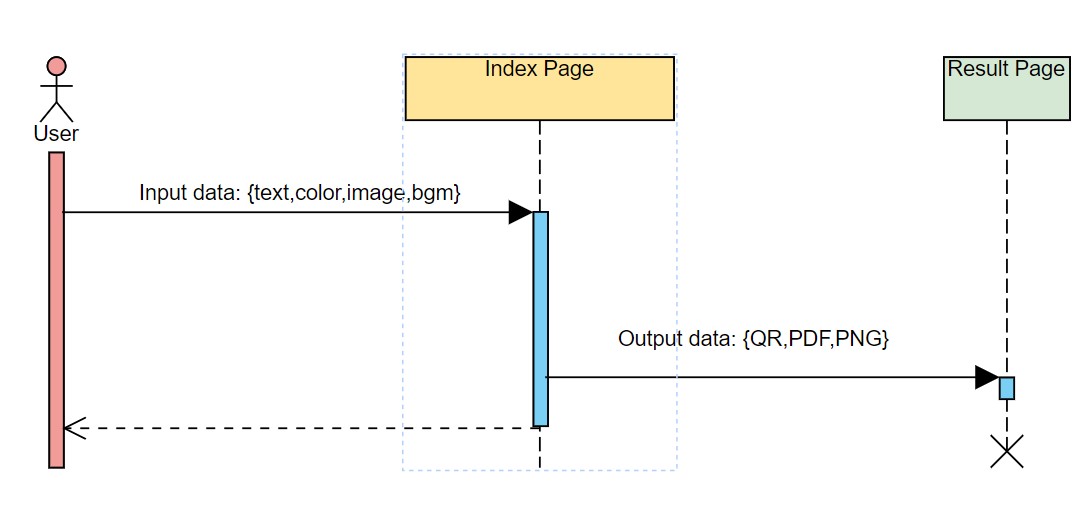


Fig - Sequence Diagram for Project

1. Deployment Diagram-

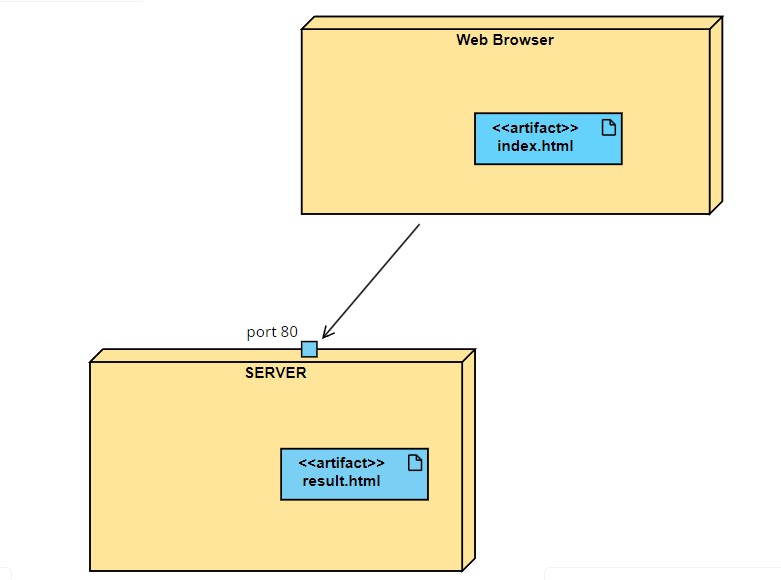
A deployment diagram in the Unified Modeling Language (UML) illustrates the physical deployment of software components in a system and their interactions with hardware and network infrastructure. 

Fig - Deployment Diagram for Project

Components:

Client (Web Browser):

* + Description: Represents the web browsers used by end-users to access the qrsite application.

Backend Server:

* + Description: The server that hosts the backend of the application, including the python and flask server handling API request

1. DataflowDiagram-

Data Flow:

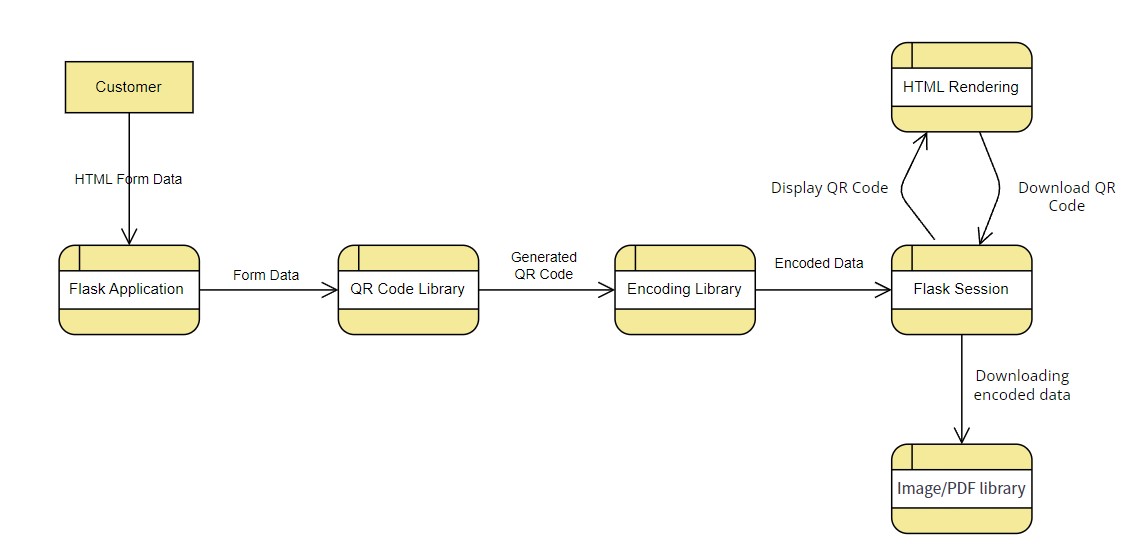
* User interacts with the web browser to input data and submit the form on the index.html page.
* The Flask app (server-side) receives the form data and processes it using the QR code library to generate a QR code image.
* If the user uploaded a logo, the QR code is overlaid with the logo.
* The app may use an image/PDF library to add a heading to the QR code if specified by the user.
* The resulting QR code image is converted to a base64-encoded string and stored in the session.
* The server sends the result.html page to the user's web browser, displaying the QR code image and allowing the user to download it as a PNG or PDF.
* If the user chooses to download, the server sends the base64-encoded string to the server, and the Flask app processes the request.
* The Flask app may use the image/PDF library to create a PDF document with the QR code and heading (if specified).
* The generated file (PNG or PDF) is sent to the user's web browser for download.

fig: Data Flow Diagram for Project