

Lab Module-1

Lab work



April 17, 2025

TOPS TECHNOLOGIC

MANINAGAR BRANCH

Q-1 Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

* <html>

<head>

<title>

HELLO WORLD

</title>

</head>

<body>

HELLO WORLD

</body>

</html>

* PYTHON

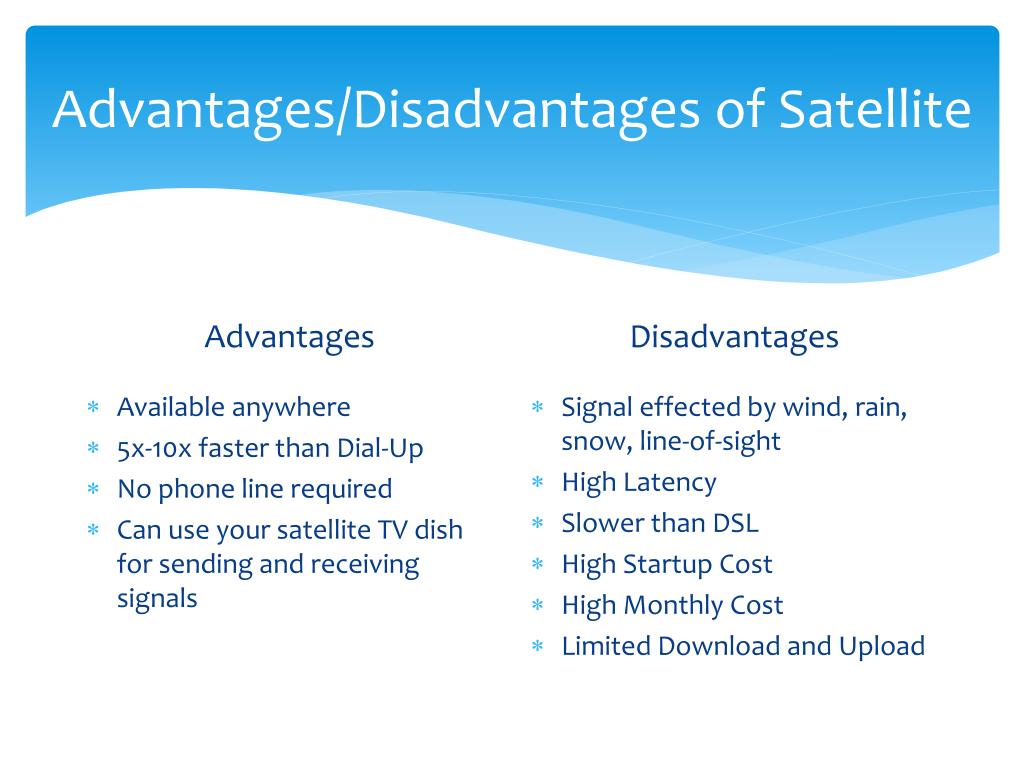
print ("Hello World!")

output: Hello World

Q-2 Research and create a diagram of how data is transmitted from a client to a server over the internet.

Q-3 Design a simple HTTP client-server communication in any language.

Q-4Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.

* 

Q-5 Simulate HTTP and FTP requests using command line tools (e.g., curl).

* 1. Simulate HTTP Request with curl
* a. GET Request
* curl http://example.com
* This sends a basic HTTP GET request to example.com.
* b. POST Request
* curl -X POST -d "username=user password=pass" http://example.com/login
* This sends a POST request with form data to a login endpoint.
* c. Custom Headers
* curl -H "Content-Type: application/Json" -H "Authorization: Bearer token123" http://example.com/api/data
* d. Download a File
* curl -O http://example.com/file.zip
* 2. Simulate FTP Request with curl
* a. Download File from FTP
* curl -u username: password ftp://ftp.example.com/file.txt -O
* b. Upload File to FTP
* curl -T upload.txt -u username: password ftp://ftp.example.com/
* c. List Files in Directory
* curl -u username: password ftp://ftp.example.com/

Q-6 Identify and explain three common application security vulnerabilities. Suggest possible solutions.

1. SQL Injection (SQLi)

Description:

SQL Injection occurs when user input is directly included in SQL queries without proper sanitization, allowing attackers to manipulate queries and access or alter database information.

Example:

SELECT \* FROM users WHERE username = '$input';

If $input is ' OR '1'='1, the query becomes:

SELECT \* FROM users WHERE username = '' OR '1'='1';

Solutions:

Use prepared statements or parameterized queries.

Validate and sanitize all user inputs.

Employ ORM tools (like Sequelize, Hibernate) which help prevent SQLi.

2. Cross-Site Scripting (XSS)

Description:

XSS occurs when an attacker injects malicious scripts into a trusted website. These scripts can execute in users' browsers, steal session cookies, or redirect them.

Types: Stored, Reflected, and DOM-based XSS.

Solutions:

Escape or encode output to prevent script execution.

Use frameworks that auto-handle escaping (like React or Angular).

Apply Content Security Policy (CSP) headers.

Validate and sanitize user input strictly.

3. Cross-Site Request Forgery (CSRF)

Description:

CSRF tricks an authenticated user into unknowingly submitting a malicious request. For example, a forged form submission that changes account settings.

Solutions:

Use anti-CSRF tokens in forms and headers.

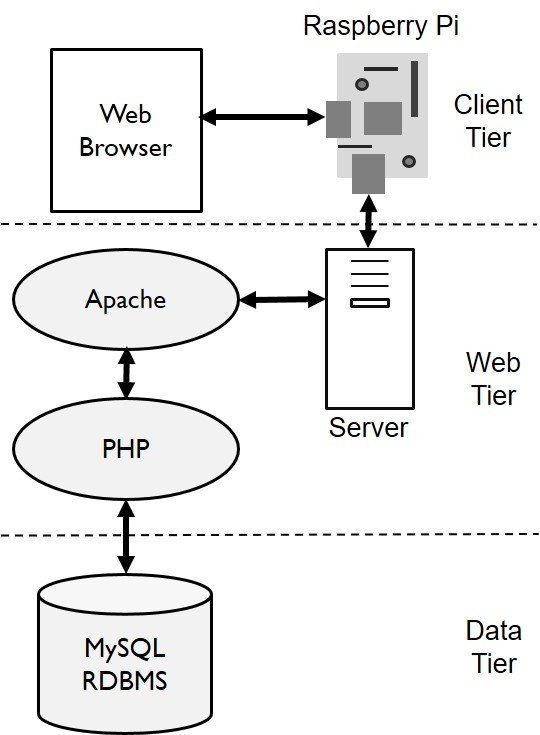
Verify request origin using Same Site cookies.

Require re-authentication for sensitive operations.

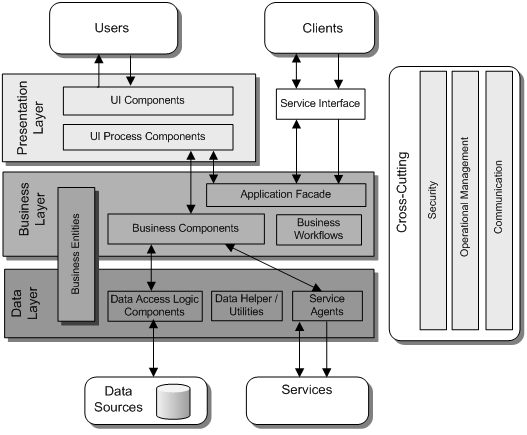
Q-7 Identify and classify 5 applications you use daily as either system software or application software.

* **Operating System (e.g., Windows, macOS, Android):**
* System Software. System software manages hardware and software resources, providing a foundation for other applications.
* **2. Web Browser (e.g., Chrome, Firefox, Safari):**
* Application Software. Application software provides a user interface for interacting with the computer and performing specific tasks, such as browsing the internet.
* **3. Word Processor (e.g., Microsoft Word, Google Docs):**
* Application Software. Application software enables users to create and edit text documents, a specific type of user-oriented task.
* **4. Email Client (e.g., Outlook, Gmail, Thunderbird):**
* Application Software. Application software facilitates sending, receiving, and managing emails, a user-oriented task.
* **5. Video Conferencing Software (e.g., Zoom, Google Meet):**
* Application Software. Application software allows users to engage in real-time communication, a user-oriented task.

Q-8 Design a basic three-tier software architecture diagram for a web application.

* 

Q-9 Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

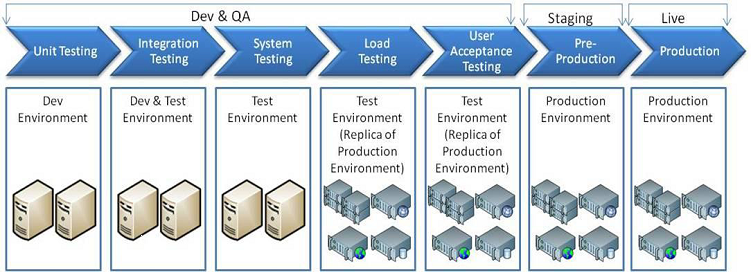


Q-10 Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.

* Software Development & Production stages :



* Environment Testing



Q-12 Write and upload your first source code file to Github.

* Step 1: Create a GitHub Repository

1. Go to https://github.com and log in.

2. Click the "+" icon on the top right > choose "New repository".

3. Enter a repository name (e.g., my-first-code).

4. Choose Public or Private.

5. (Optional) Add a README file.

6. Click "Create repository"

Step 2: Write a Simple Code File

For example, here’s a basic Python file:

Filename: hello.py

print("Hello, GitHub!")

Step 3: Upload the File

Option 1: Upload via GitHub Web UI

1. Go to your repository page.

2. Click "Add file" > "Upload files".

3. Drag your hello.py file or choose it from your computer.

4. Add a commit message (e.g., "Add hello.py").

5. Click "Commit changes".

Option 2: Upload via Git Command Line

1. Open Terminal or Git Bash.

2. Run the following commands:

git config --global user.name "Your Name"

git config --global user.email "your@email.com"

mkdir my-first-code

cd my-first-code

echo 'print("Hello, GitHub!")' > hello.py

git init

git add hello.py

git commit -m "Add hello.py"

git branch -M main

git remote add origin https://github.com/YOUR\_USERNAME/my-first-code.git

git push -u origin main

Replace YOUR\_USERNAME with your GitHub username.

Q-13 Create a Github repository and document how to commit and push code changes.

* Step 1: Create a GitHub Repository
* 1. Go to https://github.com and sign in.
* 2. Click the "+" icon in the top-right corner and select "New repository".
* 3. Fill in the details:
* Repository name: example-repo
* Description: (Optional)
* Choose Public or Private
* Optionally initialize with a README
* 4. Click Create repository
* Step 2: Set Up Git Locally
* Make sure Git is installed on your machine:
* git --version
* If not, download and install Git.
* Configure your Git identity (only needed once):
* git config --global user.name "Your Name"
* git config --global user.email "you@example.com"
* Step 3: Clone the Repository
* In your terminal, run:
* git clone https://github.com/YOUR\_USERNAME/example-repo.git
* cd example-repo
* Replace YOUR\_USERNAME with your GitHub username.
* Step 4: Create and Add a File
* Create a new file:
* echo "print('Hello, GitHub!')" > hello.py
* Add the file to staging:
* git add hello.py
* Step 5: Commit Your Changes
* Commit the staged file with a message:
* git commit -m "Add hello.py"
* Step 6: Push Changes to GitHub
* Push to the main branch:
* git push origin main
* You’ll see the file appear in your GitHub repo.

Q-14 Create a student account on Github and collaborate on a small project with a classmate.

Q-15 Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.

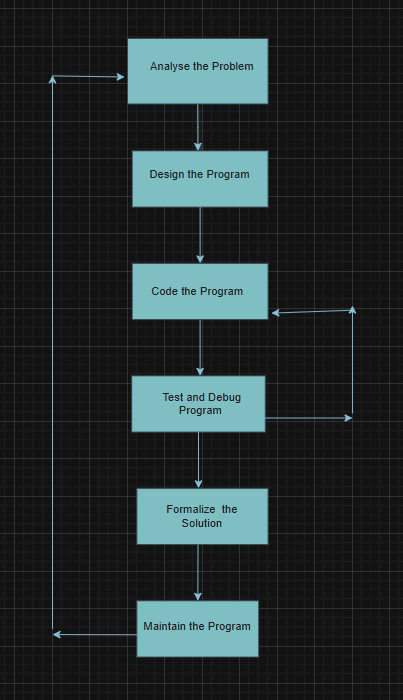
* System Software:
* **Operating System (OS):**
* Windows, macOS, Linux. These are the foundation upon which all other software runs.
* **Device Drivers:**
* Software that enables the OS to communicate with hardware devices, such as a printer or graphics card.
* **System Utilities:**
* Tools like antivirus software, file managers, and disk defragmenters. These are essential for maintaining the OS and computer performance.
* Application Software:
* **Office Suite:** Microsoft Office (Word, Excel, PowerPoint), Google Workspace (Docs, Sheets, Slides). Used for creating and editing documents, spreadsheets, presentations, and emails.
* **Web Browsers:** Chrome, Firefox, Safari. Used for accessing and navigating the internet.
* **Communication Tools:** Zoom, Teams, Skype. Facilitate communication and collaboration.
* **Media Players:** VLC, iTunes, Spotify. Used for playing audio and video files.
* **Graphics and Design Software:** Photoshop, CorelDRAW, Canva. Used for creating and editing images, graphics, and designs.
* **Games:** Fortnite, Minecraft. Entertainment and interactive software.
* Utility Software:
* **Antivirus Software:** Norton, McAfee. Protects against malware and viruses.
* **File Managers:** Windows File Explorer, Finder (macOS). Used for organizing and managing files on the computer.
* **Disk Defragmenters:** Helps optimize the storage on a hard drive.
* **System Restore:** Used to restore a system to a previous state.Q
* **Backup Software:** Tools like Time Machine (macOS) or cloud backup services. Used to create copies of data.

Q-16 Follow a GIT tutorial to practice cloning, branching, and merging repositories

Q-17 Write a report on the various types of application software and how they improve productivity.

* Application software performs specific user tasks, boosting productivity by:
* Automating repetitive actions: Example: Spreadsheet software (like Excel) automating calculations.
* Increasing efficiency and speed: Example: Word processing software (like Word) quickly formatting documents.
* Improving accuracy and reducing errors: Example: Spell-check in email clients (like Gmail).
* Enhancing organization and information management: Example: Database software (like Access) structuring customer data.
* Facilitating collaboration and communication: Example: Messaging platforms (like Slack) enabling real-time team discussions.
* Enabling data-driven decisions: Example: CRM software (like Salesforce) analyzing sales trends.
* Offering accessibility and flexibility: Example: Cloud-based document editors (like Google Docs) allowing access from any device.

Q-18 Create a flowchart representing the Software Development Life Cycle (SDLC)



Q-18 Write a requirement specification for a simple library management system.

1. Introduction

1.1. Purpose:

This document outlines the requirements for a simple Library Management System (LMS). The purpose of this system is to automate basic library operations, such as managing books, members, and book loans.

1.2. Scope:

This initial version of the LMS will focus on the core functionalities of:

\* Adding, updating, and deleting book information.

\* Registering, updating, and deactivating member accounts.

\* Allowing members to borrow and return books.

\* Tracking book availability.

1.3. Intended Audience:

This document is intended for the developers who will build the system, the stakeholders (librarians and library users) who will use it, and anyone involved in the project's planning and execution.

2. Overall Description

2.1. Product Perspective:

The LMS will be a standalone web-based application accessible through standard web browsers. It will provide a user-friendly interface for librarians and a separate interface for library members (if deemed necessary for this simple version).

2.2. User Classes and Characteristics:

\* Librarian: Responsible for managing the library's inventory (books), member accounts, and loan transactions. Requires full access to all system functionalities.

\* Library Member: (For this simple version, interaction might be limited to viewing their borrowed books, if implemented). Primarily interacts with the system for borrowing and returning books (potentially via librarian interaction).

2.3. Operating Environment:

The system should be compatible with common web browsers (Chrome, Firefox, Safari, Edge) on standard desktop and laptop computers. Specific server-side requirements (operating system, database) will be determined during the design phase.

2.4. Design and Implementation Constraints:

\* The system should be developed using commonly used and well-documented technologies.

\* Security of data (especially member information) is a concern.

\* The system should be designed for ease of use and maintainability.

2.5. Assumptions:

\* Each book will have a unique identifier (ISBN or a system-generated ID).

\* Each library member will have a unique identifier.

\* The system will not handle overdue fines in this initial version.

3. Specific Requirements

3.1. Functional Requirements:

3.1.1. Book Management:

\* FR01: Add New Book: The system shall allow librarians to add new book information, including title, author(s), ISBN, publication year, publisher, and number of copies.

\* FR02: View Book Details: The system shall allow librarians to view all details of a specific book.

\* FR03: Update Book Information: The system shall allow librarians to modify existing book information (title, author, etc.).

\* FR04: Delete Book: The system shall allow librarians to remove a book record from the system.

\* FR05: Search Books: The system shall allow librarians to search for books based on title, author, and ISBN.

\* FR06: Track Book Availability: The system shall automatically update and display the number of available copies for each book.

3.1.2. Member Management:

\* FR07: Register New Member: The system shall allow librarians to register new library members, including their name, contact information (address, phone, email).

\* FR08: View Member Details: The system shall allow librarians to view all details of a specific member.

\* FR09: Update Member Information: The system shall allow librarians to modify existing member information.

\* FR10: Deactivate Member Account: The system shall allow librarians to deactivate a member's account.

\* FR11: Search Members: The system shall allow librarians to search for members based on name and member ID.

3.1.3. Loan Management:

\* FR12: Borrow Book: The system shall allow librarians to record the borrowing of a book by a registered member. This should:

\* Verify the member's active status.

\* Check if there are available copies of the book.

\* Associate the book with the borrowing member and record the issue date.

\* Decrement the number of available copies.

\* FR13: Return Book: The system shall allow librarians to record the return of a borrowed book. This should:

\* Identify the borrowed book.

\* Update the book's status as returned.

\* Increment the number of available copies.

\* Record the return date.

\* FR14: View Borrowed Books (per Member): The system shall allow librarians to view a list of all books currently borrowed by a specific member.

\* FR15: View All Borrowed Books: The system shall allow librarians to view a list of all currently borrowed books.

3.2. Non-Functional Requirements:

3.2.1. Usability:

\* NFR01: Ease of Use: The system should have a clear and intuitive user interface for librarians.

\* NFR02: Response Time: The system should respond to user actions within a reasonable timeframe (e.g., search results displayed within 3 seconds).

3.2.2. Reliability:

\* NFR03: Availability: The system should be available during library operating hours with minimal downtime.

\* NFR04: Data Integrity: The system should ensure the accuracy and consistency of stored data.

3.2.3. Security:

\* NFR05: Librarian Authentication: The system shall require librarians to log in with valid credentials to access administrative functionalities.

\* NFR06: Data Security: Sensitive data (e.g., member contact information) should be stored securely.

3.2.4. Performance:

\* NFR07: Scalability: The system should be able to handle a reasonable increase in the number of books and members.

3.3. User Interface Requirements:

\* UI01: Web-Based Interface: The system shall have a web-based user interface accessible through standard web browsers.

\* UI02: Clear Navigation: The interface should have clear and consistent navigation for all functionalities.

\* UI03: Data Presentation: Information should be displayed in a clear and organized manner.

4. Future Enhancements (Out of Scope for this Simple Version):

\* Overdue fine management.

\* Member self-service portal for searching and viewing borrowed books.

\* Integration with barcode scanners.

\* Generating reports (e.g., popular books, member statistics).

\* Book reservation system.

5. Glossary:

\* LMS: Library Management System

\* ISBN: International Standard

Q-19 Perform a functional analysis for an online shopping system

* A functional analysis for an online shopping system involves identifying all the functions or features the system must perform to meet user needs and business objectives. These functions can be categorized into user management, product management, shopping cart and checkout, order management, and payment processing. For example, the system should allow users to register, log in, browse products, add items to a cart, initiate checkout, select payment methods, and track orders.

Here's a more detailed breakdown:

1. User Management:

* **User Registration:**

Users should be able to create accounts with necessary details (username, password, email, etc.).

* **User Login:**

Users should be able to securely log in using their credentials.

* **Profile Management:**

Users should be able to manage their profiles, including updating details, changing passwords, and viewing order history.

* **Account Security:**

Implement robust security measures to protect user data, such as password encryption and strong authentication.

2. Product Management:

* **Product Catalog:**

The system should display a wide range of products with detailed descriptions, images, and pricing.

* **Search and Filtering:**

Users should be able to search for products by keywords, categories, or other criteria.

* **Product Details:**

Each product page should provide comprehensive information, including product images, descriptions, specifications, and user reviews.

* **Product Recommendations:**

Suggest similar or related products based on user browsing history or preferences.

3. Shopping Cart and Checkout:

* **Adding to Cart:** Users should be able to add products to their shopping cart.
* **Cart Management:** Users should be able to view, edit, and delete items in their cart.
* **Checkout Process:** Users should be able to initiate the checkout process and provide shipping and billing information.
* **Payment Processing:** The system should support various payment methods and securely process transactions.
* **Order Summary:** Provide a clear summary of the order, including items, prices, shipping costs, and taxes.
* **Order Confirmation:** Send a confirmation email to the user upon placing the order.

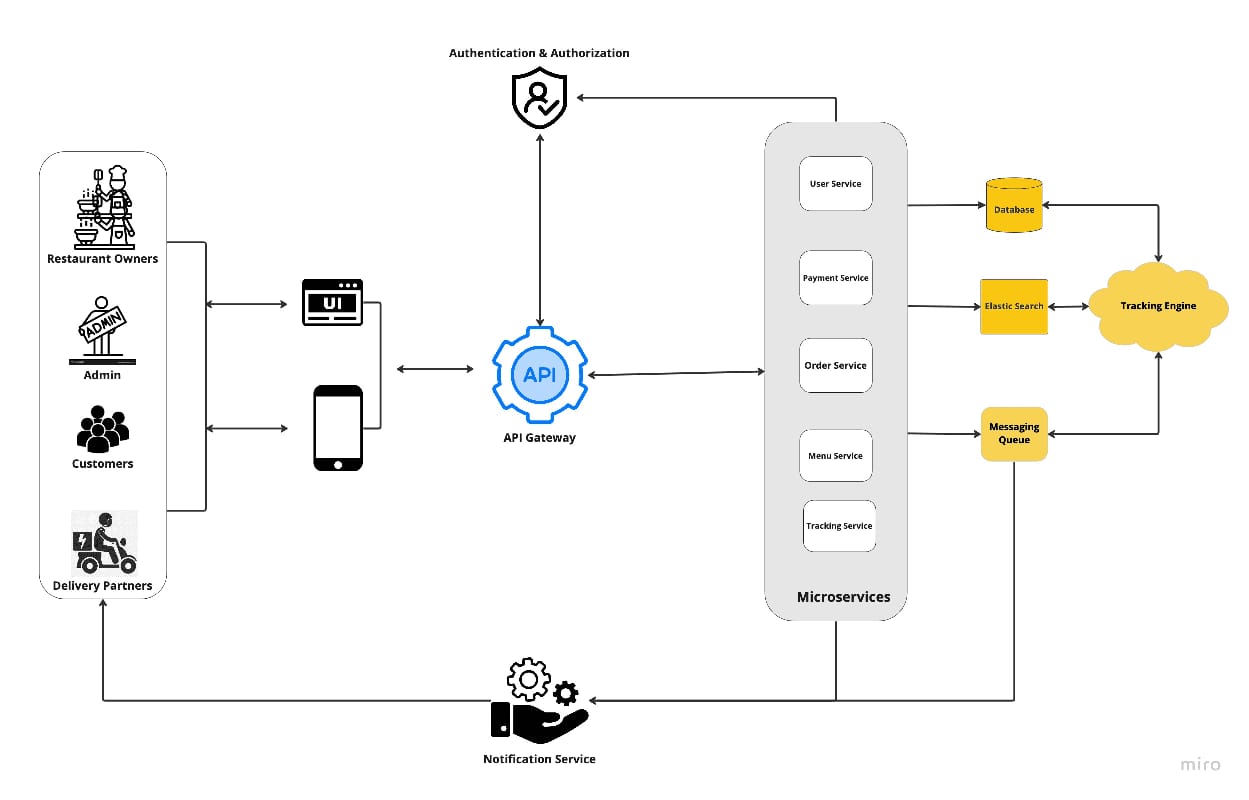
4. Order Management:

* **Order Tracking:** Users should be able to track the status of their orders.
* **Order History:** Users should have access to their order history, including past orders and associated details.
* **Order Management (Admin):** Administrators should be able to manage orders, view order details, and update order statuses.
* **Order Processing:** The system should efficiently process orders, including shipping and returns.
* **Customer Support:** Provide channels for customers to contact support regarding their orders.

5. Payment Processing:

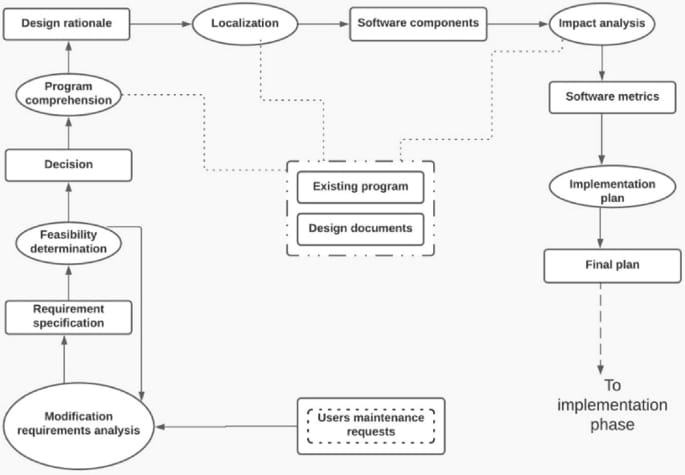
* **Payment Gateway Integration:** Integrate with secure payment gateways to process transactions.
* **Payment Method Support:** Support various payment methods, such as credit cards, PayPal, and other online payment options.
* **Fraud Prevention:** Implement measures to prevent fraudulent transactions and protect customer data.
* **Payment Confirmation:** Send confirmation to the user and the seller upon successful payment.

Q-20 Design a basic system architecture for a food delivery app.

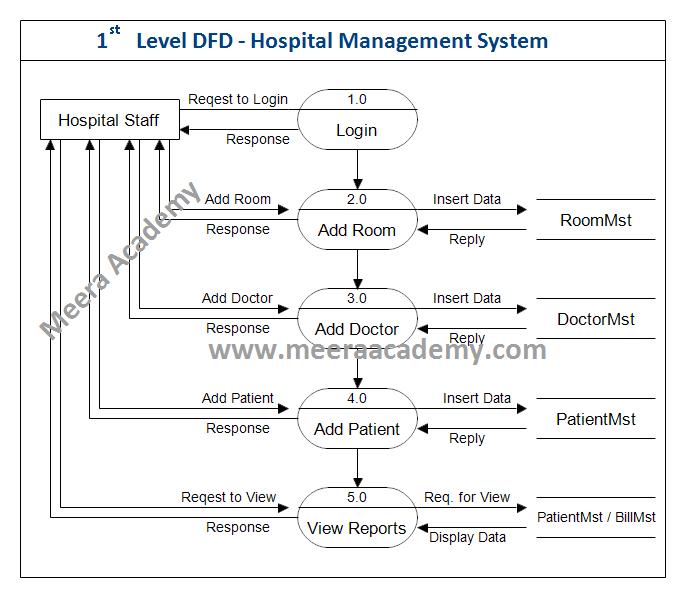


Q-21 Develop test cases for a simple calculator program.

Q-22 Document a real-world case where a software application required critical maintenance.



Q-23 Create a DFD for a hospital management system.



Q-24 Build a simple desktop calculator application using a GUI library.

* import tkinter as tk
* def click(event):
* text = event.widget.cget("text")
* if text == "=":
* try:
* result = eval(str(entry.get()))
* entry.delete(0, tk.END)
* entry.insert(tk.END, str(result))
* except Exception as e:
* entry.delete(0, tk.END)
* entry.insert(tk.END, "Error")
* elif text == "C":
* entry.delete(0, tk.END)
* else:
* entry.insert(tk.END, text)
* # Window setup
* root = tk.Tk()
* root.geometry("300x400")
* root.title("Simple Calculator")
* # Entry field
* entry = tk.Entry(root, font="Arial 20")
* entry.pack(fill=tk.BOTH, ipadx=8, pady=10, padx=10)
* # Button labels
* buttons = [
* ["7", "8", "9", "/"],
* ["4", "5", "6", "\*"],
* ["1", "2", "3", "-"],
* ["0", ".", "=", "+"],
* ["C"]
* ]
* # Create buttons
* for row in buttons:
* frame = tk.Frame(root)
* frame.pack()
* for btn in row:
* b = tk.Button(frame, text=btn, font="Arial 18", width=5, height=2)
* b.pack(side=tk.LEFT, padx=5, pady=5)
* b.bind("<Button-1>", click)
* root.mainloop()

Q-25 Draw a flowchart representing the logic of a basic online registration system