**Frontend Leb Assignment**

Module 1 - Foundation

**Q.1. Create Github account & make repository.**

**Ans.** **1.** Create a GitHub Account

* Go to the GitHub website at <https://github.com/>.
* Click the "Sign up" button.
* Enter your email address, create a password, and choose a username.
* You may be asked to solve a puzzle or verify your email address via a code sent to your email.
* Once your account is verified, you can choose to skip personalization options and go to your dashboard.

2. Create your first repository

* In the upper-right corner of any page, click the '+' icon and choose "New repository".
* Enter a name for your repository (e.g., "my-first-repo").
* Optionally, add a description for your repository.
* Choose a visibility (public or private) for your repository.
* Select "Initialize this repository with a README" to create a README file for your project description.
* Click "Create repository".

You have now created your GitHub account and your first repository.

Module 2 - Fondamentals of world wide web

**Q.1. Explain phases of SDLC life cycle.**

**Ans.** **The**[Software Development Life Cycle](https://www.google.com/search?cs=0&sca_esv=cba8e83070f3f7ab&sxsrf=AE3TifPUQHxn9k8yGqmkJFpPCwjo-e29zw%3A1754043071843&q=Software+Development+Life+Cycle&sa=X&ved=2ahUKEwiHvbvRr-mOAxUN2TgGHS2MLkoQxccNegQIBBAB&mstk=AUtExfDz0f9u3u1fyg5yaSs7oP2ZfnRP0tDwI1gZnCG1vRPRwCdKGN56ping8OSVrjMIGu83rO25jS21_zEcIrb6BcvfohCXGx9AI_TyeLEwdkGhVWrgWUOwGt1oF1-ZNwZz5tK7kkkO4B7yW4HszTl09Bdf1E5utxDF2Tc4x9oveR1rFEoN6L7SqLtXhW0nCBjQLhoQR_SleMZrqFPfaOZOVPB51DMyOXgr_t4HLnyeevHWFJp2OrKc8RN1hSKF8_GcR64tdRd1As0-Hbd8StYO95IkV28B0A4OB-nMPESPcHSEuA&csui=3) (SDLC) typically involves seven phases: Planning, Requirements Analysis, Design, Development (Coding), Testing, Deployment, and Maintenance. These phases provide a structured approach to building and maintaining software, ensuring it meets user needs and is delivered on time.

Module 3 – Fundamentals of IT

**\_What is a Program?**

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

**Ans. HTML program: "Hello World"**

HTML (HyperText Markup Language) is used to create the structure and content of a web page. It uses a tag-based syntax, where elements are enclosed in angle brackets.

html

<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

</head>

<body>

<h1>Hello World!</h1>

</body>

</html>

**\_ Explanation of the HTML structure:**

The HTML code includes the document type declaration <!DOCTYPE html>, the root <html> element, and the <head> and <body> sections. The <head> contains meta-information like the <title>, while the <body> holds the visible content, such as the <h1> heading.

**2. CSS program: Styling "Hello World"**

CSS (Cascading Style Sheets) is a stylesheet language used to define the presentation and layout of a web page. It describes how HTML elements should look.

css

*/\* style.css \*/*

body {

background-color: lightblue;

text-align: center;

}

h1 {

color: navy;

font-size: 3em; */\* 3 times the default font size \*/*

}

To apply this CSS to the HTML file, it needs to be linked. Add the following line within the <head> section of the HTML file:

html

<link rel="stylesheet" href="style.css">

**Explanation of the CSS structure:**

CSS rules consist of a selector (like body or h1) and a declaration block. The declaration block contains property-value pairs (e.g., background-color: lightblue;) that define the styling.

**\_** **World Wide Web & How Internet Works**

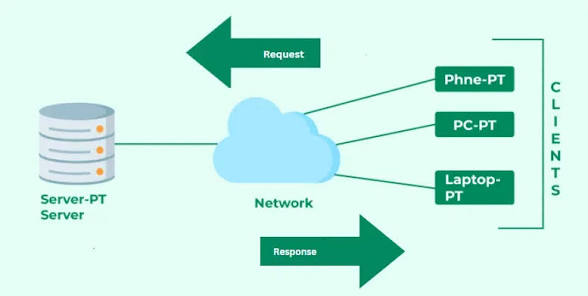
**Q.2. Research and create a diagram of how data is transmitted from a client to a server over the internet.**

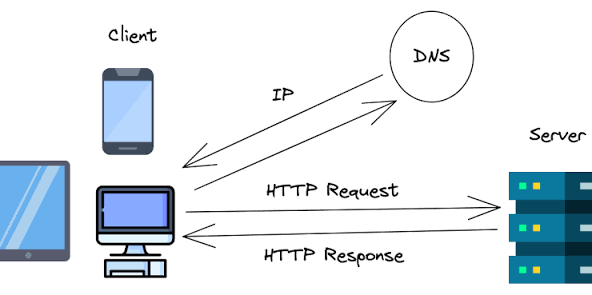
**Ans.**

**Explanation of the steps**

1. **Client Initiates Request:** The process begins when a user on a client device (like a web browser or a mobile app) initiates a request, such as typing a website address or clicking a link.
2. **DNS Lookup:** If the request involves a domain name (like "example.com"), the client sends a request to a Domain Name System (DNS) server to translate the human-readable domain name into an IP address that computers understand.
3. **DNS Resolution:** The DNS server resolves the domain name and returns the corresponding IP address to the client.
4. **Connection Establishment (TCP/IP Handshake):** The client, using the resolved IP address, attempts to establish a connection with the server. This typically involves a TCP/IP three-way handshake.
5. **Data Packetization:** Before transmission, the data (e.g., the HTTP request) is broken down into smaller units called packets.
6. **Packet Routing:** These packets are then sent through the client's local network (Wi-Fi or Ethernet) to a router or gateway, which then forwards them to the client's Internet Service Provider (ISP).
7. **Internet Backbone and Routing:** The ISP sends the packets over the internet backbone, which consists of high-speed data transmission lines connecting major networks globally. Routers along the way use the destination IP address in each packet to determine the optimal path for transmission.
8. **Server's Network and Processing:** The packets eventually arrive at the server's ISP and then to the server's local network and the server itself.
9. **Server Processes Request:** The server receives and processes the request. This may involve retrieving information from a database, performing computations, or accessing files.
10. **Server Prepares Response:** Once the server finishes processing, it prepares a response (e.g., the requested webpage, data, or file).
11. **Response Packetization and Routing:** Similar to the request, the response is broken into packets and sent back through the internet backbone, following a similar path in reverse.
12. **Client Receives and Reassembles:** The client receives the packets, reassembles them using the sequence information in the packet headers, and interprets the data to display the content or complete the requested action.

This process highlights the core principles of client-server communication over the internet, emphasizing the role of protocols like TCP/IP in ensuring reliable and efficient data transfer.





**Network Layers on Client and Server**

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

**Ans.** Designing a simple HTTP client-server communication using React primarily involves creating a React application as the client and a separate backend server (e.g., Node.js with Express) to handle requests.

**1. Server-Side (Node.js with Express):**

This example demonstrates a basic server that responds to a GET request.

// server.js

const express = require('express');

const cors = require('cors'); // For handling Cross-Origin Resource Sharing

const app = express();

const port = 5000;

app.use(cors()); // Enable CORS for all origins (for development)

app.get('/api/message', (req, res) => {

res.json({ message: 'Hello from the server!' });

});

app.listen(port, () => {

console.log(`Server listening at http://localhost:${port}`);

});

**2. Client-Side (React):**

This React component fetches data from the server and displays it.

// src/App.js

import React, { useState, useEffect } from 'react';

function App() {

const [message, setMessage] = useState('');

const [error, setError] = useState(null);

useEffect(() => {

fetch('http://localhost:5000/api/message')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.then(data => setMessage(data.message))

.catch(error => {

console.error('Error fetching data:', error);

setError('Failed to fetch message from server.');

});

}, []);

return (

<div>

<h1>Client-Server Communication</h1>

{error && <p style={{ color: 'red' }}>{error}</p>}

{message ? <p>Server message: {message}</p> : <p>Loading message...</p>}

</div>

);

}

export default App;

**Explanation:**

* **Server:**

The Node.js server uses express to create an API endpoint /api/message. It listens for GET requests and responds with a JSON object containing a "message" property. cors is used to allow the React client (running on a different port) to access the server.

* **Client:**

The React App component uses useState to manage the message and error states. The useEffect hook performs the data fetching when the component mounts.

* + fetch('http://localhost:5000/api/message') makes an HTTP GET request to the server.
  + The .then() blocks handle the successful response, parsing the JSON and updating the message state.
  + The .catch() block handles any errors during the fetch operation, updating the error state.
* **Communication:**

When the React application loads, it sends a request to the Node.js server. The server processes the request and sends back a response, which the React application then displays.

**Types of Internet Connections**

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

**Ans.** Fiber optic connections provide the fastest and most reliable speeds, but may be limited in availability and more expensive.

1. **Fiber Optic Internet:**

* **Pros:**
  + **Extremely fast speeds:** Fiber optic cables transmit data using light pulses, allowing for much faster speeds than other connection types.
  + **High bandwidth:** Fiber connections can handle large amounts of data transfer, making them ideal for streaming, gaming, and large downloads.
  + **Low latency:** Data travels quickly with minimal delay, making it suitable for real-time applications like online gaming and video conferencing.
  + **Reliable and consistent speeds:** Fiber connections are less susceptible to interference and offer more stable speeds compared to other technologies.
* **Cons:**
  + **Limited availability:** Fiber optic infrastructure is not yet available everywhere, especially in rural areas.
  + **Higher cost:** Fiber optic installations and monthly plans can be more expensive than other options.
  + **Installation challenges:** Fiber optic cables are delicate, and installation can be complex and potentially costly, especially in challenging terrains.

**2. Cable Internet:**

* **Pros:**
  + **Good speeds and bandwidth:** Cable internet offers decent speeds and is suitable for most households, though not as fast as fiber.
  + **Widely available:** Cable internet infrastructure is widely available in many areas.
  + **Good value:** Cable internet is often a good balance between speed, cost, and availability.
* **Cons:**
  + **Slower than fiber:** Cable internet speeds can be slower than fiber optic connections, especially during peak usage times.
  + **Latency issues:** Cable can experience higher latency than fiber, which may be noticeable in online gaming or video conferencing.
  + **Susceptible to interference:** Cable connections can be affected by signal interference from other devices or environmental factors.

**3. DSL (Digital Subscriber Line):**

* **Pros:**
  + **Widely available:** DSL is available in many areas, especially in more rural locations.
  + **Affordable:** DSL is often a more affordable option than fiber or cable.
* **Cons:**
  + **Slow speeds:** DSL speeds are typically much slower than fiber or cable, with limited bandwidth.
  + **Latency issues:** DSL can experience high latency, which can affect real-time online activities.
  + **Distance-sensitive:** DSL speeds decrease with distance from the provider's central office.

**4. Satellite Internet:**

* **Pros:**
  + **Availability in remote areas:** Satellite internet is an option for locations where other connection types are unavailable.
  + **Relatively easy setup:** Satellite dishes can be installed in various locations.
* **Cons:**
  + **High latency:** Satellite internet has high latency due to the distance data travels to and from satellites.
  + **Susceptible to weather:** Rain, snow, and other weather conditions can disrupt satellite internet signals.
  + **Slower speeds than other options:** Satellite internet speeds are generally slower than fiber, cable, or even DSL.
  + **Costly:** Satellite internet can be more expensive than other options, especially for higher speeds and unlimited data plans.

**5. 5G Home Internet:**

* **Pros:**
  + **Good speeds:** 5G home internet can offer decent speeds, comparable to cable or DSL, in some areas.
  + **Wireless and portable:** Offers a convenient wireless connection without needing a physical line.
* **Cons:**
  + **Variable speeds:** 5G speeds can fluctuate depending on network congestion and distance from the cell tower.

**Protocols**

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

**Ans.** Command-line tools like curl and wget can be used to simulate HTTP and FTP requests.

**Simulating HTTP Requests with curl:**

GET Request (retrieve content).

curl <https://www.example.com>

This command retrieves the HTML content of https://www.example.com and displays it in the terminal. POST Request (send data).

curl -X POST -d "param1=value1&param2=value2" <https://www.example.com/api/submit>

This sends a POST request with the specified data to the given URL. The -X POST explicitly sets the HTTP method, and -d provides the data to be sent. HEAD Request (fetch headers only).

curl -I <https://www.example.com>

The -I option instructs curl to only retrieve the HTTP headers, not the body of the response. Downloading a file.

curl -O <https://www.example.com/path/to/file.zip>

The -O option saves the downloaded content to a local file with the same name as the remote file.

Simulating FTP Requests with curl:

* **Downloading a file from an FTP server:**

curl -u "username:password" <ftp://ftp.example.com/path/to/file.txt>

This downloads file.txt from the specified FTP server using the provided username and password. The -u option is used for authentication.

* **Uploading a file to an FTP server:**

curl -u "username:password" -T /local/path/to/upload.txt <ftp://ftp.example.com/remote/path/upload.txt>

The -T option specifies the local file to upload, and the URL indicates the target location on the FTP server.

**Application Security**

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

**Ans.** SQL injection, Cross-Site Scripting (XSS), and insecure direct object references (IDOR).

1. [SQL Injection](https://www.google.com/search?rlz=1C1PNBB_enIN1170IN1170&cs=0&sca_esv=3d1e29cf7e253e0e&sxsrf=AE3TifNvYg16TvzrOAdBhuGBh9hm1pPOdg%3A1754071947562&q=SQL+Injection&sa=X&ved=2ahUKEwiByaaam-qOAxU6VzABHaP4OMQQxccNegQIChAC&mstk=AUtExfBxnTkjjH1XTx4PJlIX2K4Mnwcem4sW5y8ORyiYIS9VFIJNWSCgDgLfcnWi0biqJMSAEtyvDwPrslWTMa-yx_0VGnAEqNN7ojpZ5MMyl6PfW4BgZt6Vf_pt7Ugb70xRzLCBgzPW68dir-Grf-zmTL_QUxmWfpkv7b9sUQZ2rW3KpiXa30skz8eqnWFyKSiMlP8UVWQCXrx4x3GvNX2mEotkjHfVJlbgvMp4YvGz_9LZIjfkC6M9NY1hlSWC_dGqFfOIWlRxKYTJep5I7Zq6hyLBKiwP15MpenywGBXRZoGBGxTbVIsg0whuvpitsi_RQeyOc7UNxAjhG_E_q63pJtdYCg_DlFjFHadhPkPFUnB7ZtY--MV8BfxH7-ut1JeP20y4A-OQ9LAe9rwjKfRExXGsPXjTetDbTAycLVNdQxDI1-XMPHgsCyACYmxZy92Un9b2s5sUOwArdhFnem8Qnb1l5PueZtF4nwa4Ntq_Xg&csui=3):

* **What it is:**

SQL injection occurs when an attacker inserts malicious SQL code into input fields, which is then executed by the application's database. This can lead to unauthorized data access, modification, or even complete control of the database.

* **Example:**

An attacker might enter ' OR '1'='1 into a username field, which could bypass authentication checks if the application doesn't handle such input correctly, [according to Legit Security](https://www.legitsecurity.com/aspm-knowledge-base/application-vulnerability-common-types).

* **Possible solutions:**
  + **Parameterized queries:** Use parameterized queries or prepared statements to separate user input from the SQL code, preventing attackers from injecting malicious SQL.
  + **Input validation:** Validate all user inputs to ensure they conform to expected formats and data types.
  + **Least privilege:** Grant the application's database user the minimum necessary privileges.

2. Cross-Site Scripting (XSS):

* **What it is:**

XSS vulnerabilities allow attackers to inject malicious scripts (usually JavaScript) into web pages viewed by other users. This can lead to various attacks, such as stealing user cookies, redirecting users to malicious sites, or defacing websites.

* **Example:**

An attacker might inject a script that steals session cookies when a user views a page with a vulnerable comment section.

* **Possible solutions:**
  + **Output encoding:** Encode user-provided data before displaying it on the page, preventing it from being interpreted as code.
  + **Input validation:** Validate user input to ensure it doesn't contain potentially malicious script elements.
  + **Content Security Policy (CSP):** Implement CSP headers to restrict the sources from which a browser can load resources, limiting the impact of XSS attacks.

3. [Insecure Direct Object References (IDOR)](https://www.google.com/search?rlz=1C1PNBB_enIN1170IN1170&cs=0&sca_esv=3d1e29cf7e253e0e&sxsrf=AE3TifNvYg16TvzrOAdBhuGBh9hm1pPOdg%3A1754071947562&q=Insecure+Direct+Object+References+%28IDOR%29&sa=X&ved=2ahUKEwiByaaam-qOAxU6VzABHaP4OMQQxccNegQIUxAC&mstk=AUtExfBxnTkjjH1XTx4PJlIX2K4Mnwcem4sW5y8ORyiYIS9VFIJNWSCgDgLfcnWi0biqJMSAEtyvDwPrslWTMa-yx_0VGnAEqNN7ojpZ5MMyl6PfW4BgZt6Vf_pt7Ugb70xRzLCBgzPW68dir-Grf-zmTL_QUxmWfpkv7b9sUQZ2rW3KpiXa30skz8eqnWFyKSiMlP8UVWQCXrx4x3GvNX2mEotkjHfVJlbgvMp4YvGz_9LZIjfkC6M9NY1hlSWC_dGqFfOIWlRxKYTJep5I7Zq6hyLBKiwP15MpenywGBXRZoGBGxTbVIsg0whuvpitsi_RQeyOc7UNxAjhG_E_q63pJtdYCg_DlFjFHadhPkPFUnB7ZtY--MV8BfxH7-ut1JeP20y4A-OQ9LAe9rwjKfRExXGsPXjTetDbTAycLVNdQxDI1-XMPHgsCyACYmxZy92Un9b2s5sUOwArdhFnem8Qnb1l5PueZtF4nwa4Ntq_Xg&csui=3):

* **What it is:**

IDOR vulnerabilities occur when an application exposes internal objects (like database records or files) through direct references in URLs or other parameters, without proper access control checks. An attacker can then manipulate these references to access resources they shouldn't have access to, says Legit Security.

* **Example:**

A user might be able to access another user's profile by changing the ID in a URL, such as /profile?id=123 to /profile?id=456, if there is no proper authorization check.

* **Possible solutions:**
  + **Implement access control:** Ensure that all requests to access resources are properly authorized and that users only have access to the resources they are permitted to see.
  + **Indirect references:** Instead of direct references, use indirect references (e.g., using tokens or randomly generated IDs) that cannot be easily guessed or manipulated.
  + **Least privilege:** Ensure that users only have the minimum necessary access to resources.

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

**Ans.** Here's a breakdown of five applications used daily, classified as either system or application software:

**System Software:**

1. **. Operating System (e.g., Windows, macOS, Android):**

This is the core software that manages computer hardware and software resources and provides common services for computer programs. It's fundamental for the computer to function at all.

1. **2. Device Drivers (e.g., graphics card driver, printer driver):**

These programs allow the operating system to interact with specific hardware devices. They translate commands from the OS into instructions the hardware can understand, enabling the hardware to function correctly.

**Application Software:**

1. **Web Browser (e.g., Google Chrome, Firefox, Safari):**These programs allow users to access and interact with the internet, including websites, web applications, and other online content.
2. **Email Client (e.g., Outlook, Gmail, Thunderbird):**These applications allow users to send, receive, and manage emails.
3. **Word Processor (e.g., Microsoft Word, Google Docs):** These tools are used for creating, editing, and formatting text documents.

**Software Architecture**

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

**Ans**. A basic three-tier software architecture for a web application separates the application into three logical and physical computing tiers:

* **Presentation Tier (Client Tier):**
  + Purpose: Handles the user interface and user interaction.
  + Components: Web browser (HTML, CSS, JavaScript), mobile application.
  + Functionality: Displays information to the user, captures user input, and sends requests to the Application Tier.
* **Application Tier (Business Logic Tier / Middle Tier):**
  + Purpose: Processes user requests, executes business logic, and acts as an intermediary between the Presentation Tier and the Data Tier.
  + Components: Web server (e.g., Apache, Nginx), application server (e.g., Node.js, Java Spring Boot, Python Django/Flask), APIs.
  + Functionality: Receives requests from the Presentation Tier, applies business rules, interacts with the Data Tier to retrieve or store data, and sends responses back to the Presentation Tier.
* **Data Tier (Database Tier):**
  + Purpose: Stores and manages application data.
  + Components: Database server (e.g., MySQL, PostgreSQL, MongoDB, SQL Server), data storage (e.g., file system, cloud storage).
  + Functionality: Provides secure and efficient data storage and retrieval, manages data integrity, and handles data access requests from the Application Tier.

**Layers in Software Architecture**

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

**Ans.** A case study on the functionality of the presentation, business logic, and data access layers within a software system, exemplified by an e-commerce application, demonstrates the principles of layered architecture.

**System Overview:** The e-commerce application facilitates online shopping, including product browsing, order placement, and customer management.

**1. Presentation Layer:**

* **Functionality:**

This layer is responsible for rendering the user interface and handling user interactions. In the e-commerce application, it includes web pages for product display, shopping cart, checkout, and user accounts. It translates user input (e.g., button clicks, form submissions) into requests for the business logic layer and displays responses from the business logic layer to the user.

* **Example:**

When a user navigates to a product page, the presentation layer retrieves product details from the business logic layer and dynamically displays them using HTML, CSS, and JavaScript. When a user adds an item to the cart, the presentation layer captures this action and sends a request to the business logic layer.

**2. Business Logic Layer:**

* **Functionality:**

This layer encapsulates the core business rules and processes of the application. It acts as an intermediary between the presentation and data access layers, ensuring data integrity and enforcing business policies.

* **Example:**

In the e-commerce application, the business logic layer handles tasks such as:

* + Product Management: Validating product availability, calculating discounts, and managing inventory.
  + Order Processing: Validating order details, calculating total prices, and managing order status transitions.
  + User Management: Authenticating users, managing user profiles, and handling password resets.
  + Communication: It receives requests from the presentation layer, processes them according to business rules, and then interacts with the data access layer to retrieve or store necessary data. After data operations, it prepares the response for the presentation layer.

**3. Data Access Layer:**

* **Functionality:**

This layer is responsible for abstracting the underlying data storage mechanisms and providing a consistent interface for the business logic layer **to** interact with data. It handles all operations related to data persistence, retrieval, and modification.

* **Example:**

In the e-commerce application, the data access layer interacts with the database (e.g., SQL database) to perform operations such as:

* + Retrieving product information.
  + Storing and updating customer details.
  + Managing order records and associated items.
  + Handling database connections, queries, and error handling, shielding the business logic layer from the complexities of direct database interaction.

**Inter-Layer Communication Flow:**

* A user interacts with the Presentation Layer (e.g., clicks "Add to Cart").
* The Presentation Layer sends a request to the Business Logic Layer.
* The Business Logic Layer processes the request, applies business rules (e.g., checks stock), and interacts with the Data Access Layer to update the shopping cart in the database.
* The Data Access Layer performs the database operation and returns the result to the Business Logic Layer.
* The Business Logic Layer prepares the response and sends it back to the Presentation Layer.
* The Presentation Layer updates the user interface to reflect the changes (e.g., displays item in cart).

**Software Environments**

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

**Ans. 1. Development Environment**

* **Purpose:** Coding and testing on your local machine.
* **Tools:**
  + Node.js + npm/yarn
  + React Developer Tools
  + IDEs (VS Code)
  + Hot reload using npm start
* **Features:**
  + Verbose logs
  + Mock APIs / local servers
  + .env.development file for local config

**2. Testing Environment**

* **Purpose:** Simulate production-like conditions for QA.
* **Tools:**
  + Jest / React Testing Library
  + Cypress (for end-to-end testing)
  + ESLint / Prettier
* **Features:**
  + Continuous Integration (CI)
  + Tests run automatically
  + .env.test configuration

**3. Production Environment**

* **Purpose:** Live app users interact with.
* **Tools:**
  + React build output (npm run build)
  + Hosted on Vercel, Netlify, or custom server (e.g., Nginx)
* **Features:**
  + Optimized/minified code
  + .env.production for secure keys
  + Monitoring/logging (Sentry, etc.)

**🛠️ Set Up a React.js Development Environment in a VM**

We’ll use VirtualBox + Ubuntu and install a full React.js development setup.

**✅ Step-by-Step Setup in Ubuntu VM**

**1. Prepare the Virtual Machine**

* Install VirtualBox.
* Download Ubuntu Desktop ISO from <https://ubuntu.com/download>.
* Create a VM with at least:
  + 2 GB RAM
  + 20 GB disk
  + Ubuntu 64-bit

**2. Install Node.js and npm**

Open the terminal and run:

sudo apt update

sudo apt install nodejs npm -y

node -v

npm -v

✅ Or install latest version via NodeSource:

curl -fsSL https://deb.nodesource.com/setup\_20.x | sudo -E bash -

sudo apt install -y nodejs

**3. Install create-react-app**

sudo npm install -g create-react-app

**4. Create a New React Project**

npx create-react-app my-react-app

cd my-react-app

npm start

* This runs a development server at http://localhost:3000/

**5. Set Up Environment Files**

Inside your React project:

* .env.development
* .env.test
* .env.production

**Example:**

env

REACT\_APP\_API\_URL=http://localhost:4000

6. Add Testing Tools

Install React Testing Library:

npm install --save-dev @testing-library/react

npm install --save-dev jest

**🔍 Optional: Serve Production Build**

npm run build

npm install -g serve

serve -s build

**Source Code**

**Q.11. Write and upload your first source code file to Github.**

**Ans.**

**Step 1: Create a Simple React Component**

1. In your React project folder (e.g., my-react-app/src/), create a file named:  
   HelloWorld.js
2. Add the following code:

// HelloWorld.js

import React from 'react';

const HelloWorld = () => {

return <h1>Hello, GitHub!</h1>;

};

export default HelloWorld;

1. Use it in App.js:

// App.js

import React from 'react';

import HelloWorld from './HelloWorld';

function App() {

return (

<div className="App">

<HelloWorld />

</div>

);

}

export default App;

**Step 2: Initialize Git (if not already done)**

Open your terminal inside the project directory:

git init

git add .

git commit -m "Initial commit with HelloWorld component"

**Step 3: Push to GitHub**

**A. Create a new GitHub repository:**

1. Go to <https://github.com>
2. Click **"New"** to create a repository.
3. Give it a name (e.g., my-react-app), then click **Create Repository**.

**B. Connect your local project to the GitHub repo:**

git remote add origin https://github.com/YOUR\_USERNAME/my-react-app.git

git branch -M main

git push -u origin main

**Github and Introductions**

**Q.12. Create a Github repository and document how to commit and push code changes.**

**Ans. 1. Create a GitHub Repository**

1. Go to <https://github.com> and log in.
2. Click the "+" icon in the top-right corner → "New repository".
3. Fill out:
   * Repository name: my-react-app (or your choice)
   * Description: "First React.js App"
   * Visibility: Public or Private
   * DO NOT check Initialize with README
4. Click Create repository.

**✅ 2. Connect Local Project to GitHub**

Open a terminal inside your project directory.

If your project is not yet a Git repo:

git init

Add remote origin (replace your GitHub username):

git remote add origin https://github.com/YOUR\_USERNAME/my-react-app.git

**✅ 3. Stage, Commit, and Push Code**

**Step 1: Stage changes**

git add .

. stages all modified, new, or deleted files.

**Step 2: Commit the changes**

git commit -m "Initial commit with HelloWorld component"

**Step 3: Push to GitHub**

git branch -M main # Ensure branch is named 'main'

git push -u origin main

**✅ 4. Document Future Code Changes**

Whenever you change or add new files:

**A. Check file status**

git status

**B. Stage files**

git add .

**C. Commit with a message**

git commit -m "Added new feature or fixed bug"

**D. Push to GitHub**

git push

**Student Account in Github**

**Q.13. Create a student account on Github and collaborate on a small project with a classmate.**

**Ans.**

**1. Create a GitHub Student Account**

1. Go to: https://education.github.com/pack
2. Click "Get Student Benefits".
3. Sign in or create a GitHub account.
4. Verify with:
   * School-issued email or
   * Upload student ID / proof of enrollment
5. Wait for approval (can take a few hours or days).

**✅ 2. Create a Project Repository**

1. Log in to GitHub.
2. Click New repository.
3. Name it (e.g., student-collab-project).
4. Add description and initialize with README (optional).

**✅ 3. Add a Classmate as Collaborator**

1. Go to the repository → Settings → Collaborators.
2. Click "Add people".
3. Enter your classmate’s GitHub username and send invite.
4. They must accept the invite to access the repo.

**✅ 4. Collaborate on Code**

1. **Both students clone the repo:**

git clone https://github.com/your-username/student-collab-project.git

1. **Make changes, then:**

git add .

git commit -m "Your change message"

git push

1. **Use branches and pull requests for larger changes.**

**Types of Software**

**Q.14 Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.**

**Ans.** Here's an example of a simple HTML page that lists and classifies the software you use regularly into System, Application, and Utility software categories:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Software Classification</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

h1 {

color: #2c3e50;

}

h2 {

color: #34495e;

}

ul {

list-style-type: square;

}

.category {

margin-bottom: 20px;

}

</style>

</head>

<body>

<h1>Software Classification</h1>

<div class="category">

<h2>System Software</h2>

<ul>

<li>Microsoft Windows 10</li>

<li>Ubuntu Linux</li>

<li>macOS</li>

<li>Device Drivers (NVIDIA, Intel)</li>

</ul>

</div>

<div class="category">

<h2>Application Software</h2>

<ul>

<li>Google Chrome</li>

<li>Microsoft Word</li>

<li>Visual Studio Code</li>

<li>Adobe Photoshop</li>

<li>Zoom</li>

</ul>

</div>

<div class="category">

<h2>Utility Software</h2>

<ul>

<li>WinRAR / 7-Zip</li>

<li>Antivirus (e.g., Windows Defender, Avast)</li>

<li>CCleaner</li>

<li>Disk Management Tool</li>

<li>Backup and Restore</li>

</ul>

</div>

</body>

</html>

**GIT and GITHUB Training**

**Q.15. Follow a GIT tutorial to practice cloning, branching, and merging repositories.**

**Ans.**

**Prerequisites**

* Install Git: <https://git-scm.com/downloads>
* Have a GitHub account
* Set up Git with your info (run once):

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

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

**1. Clone a Repository**

You can practice with your own or a demo GitHub repository.

**A. Create a test repo on GitHub:**

* Go to <https://github.com/new>
* Name it something like git-practice
* Make it public or private and click **Create Repository**

**B. Clone the repo:**

git clone https://github.com/YOUR\_USERNAME/git-practice.git

cd git-practice

**2. Create and Switch to a New Branch**

git checkout -b feature-branch

**3. Make Changes and Commit**

Create a new file or edit an existing one:

echo "This is a test file" > test.txt

git add test.txt

git commit -m "Added test file in feature branch"

**4. Switch Back to Main Branch**

git checkout main

**5. Merge the Feature Branch**

git merge feature-branch

If no conflicts, the content is now merged into the main branch.

**6. Push Changes to GitHub**

git push origin main

If the branch doesn't exist on GitHub yet:

git push --set-upstream origin feature-branch

**Application Software**

**Q.16. Write a report on the various types of application software and how they improve productivity.**

**Ans.** Application software refers to programs designed to perform specific tasks for users. Unlike system software, which manages hardware, application software helps users complete practical, productive, and creative tasks. These applications play a critical role in personal, academic, and professional environments by automating, simplifying, or enhancing work processes.

**Types of Application Software**

**1. Word Processing Software**

* Examples: Microsoft Word, Google Docs, LibreOffice Writer
* Purpose: Create, edit, and format textual documents.
* Productivity Impact: Enables quick document creation with features like spell check, templates, and collaborative editing.

**2. Spreadsheet Software**

* Examples: Microsoft Excel, Google Sheets, LibreOffice Calc
* Purpose: Manage, calculate, and analyze numerical data using rows, columns, and formulas.
* Productivity Impact: Automates calculations, supports data analysis, and helps visualize data with charts and graphs.

**3. Presentation Software**

* Examples: Microsoft PowerPoint, Google Slides, Keynote
* Purpose: Create slide-based visual presentations.
* Productivity Impact: Assists in communicating ideas clearly during meetings, lectures, and pitches.

**4. Database Management Software (DBMS)**

* Examples: Microsoft Access, MySQL, Oracle Database
* Purpose: Store, retrieve, and manage structured data.
* Productivity Impact: Organizes large data sets efficiently and supports business decisions through querying and reporting.

**5. Email and Communication Software**

* Examples: Microsoft Outlook, Gmail, Slack, Microsoft Teams
* Purpose: Facilitate communication through email, messaging, and video conferencing.
* Productivity Impact: Speeds up communication, supports collaboration, and improves team coordination.

**6. Graphics and Multimedia Software**

* Examples: Adobe Photoshop, Canva, VLC Media Player
* Purpose: Create and manipulate images, videos, and audio files.
* Productivity Impact: Enhances visual content creation and supports media-rich communication.

**7. Project Management Software**

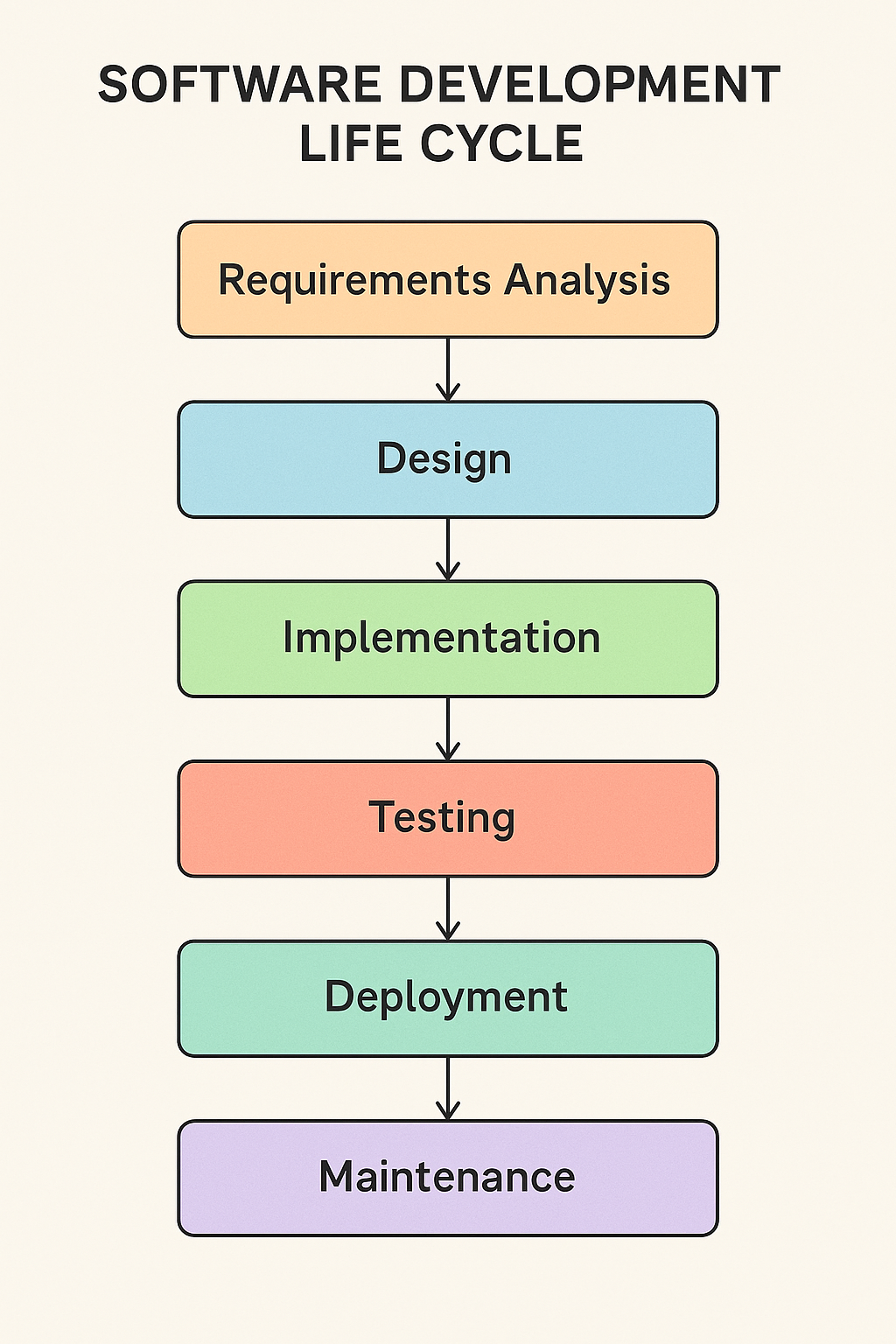
* Examples: Trello, Asana, Jira, Microsoft Project
* Purpose: Plan, track, and manage projects and tasks.
* Productivity Impact: Improves task organization, time management, and team collaboration.

**8. Web Browsers**

* Examples: Google Chrome, Mozilla Firefox, Microsoft Edge
* Purpose: Access and browse the internet.
* Productivity Impact: Enables access to cloud apps, online research, and web-based tools.

**Software Development Process**

**Q.17. Create a flowchart representing the Software Development Life Cycle (SDLC).**

**Ans. **

**Software Requirement**

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

**Ans. Requirement Specification Document**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to outline the functional and non-functional requirements for the development of a Simple Library Management System (LMS). This system will help librarians manage books, members, issue/return processes, and generate reports.

**1.2 Scope**

This LMS will provide an interface for librarians to:

* Add/edit/delete book and member records
* Issue and return books
* Track due dates and late returns
* Generate simple reports

**1.3 Definitions**

* Librarian: User who manages the library system
* Member: Person who borrows books
* ISBN: Unique book identifier
* Overdue: Book not returned by the due date

**2. Overall Description**

**2.1 Product Perspective**

This system is a standalone web/desktop application. It is designed for small libraries and schools.

**2.2 User Classes and Characteristics**

* Admin/Librarian: Can manage books, users, and transactions
* Member (optional access): Can view book availability (if public portal is enabled)

**2.3 Assumptions and Dependencies**

* The system will run on Windows/Linux
* A relational database will be used (e.g., MySQL or SQLite)

**3. Functional Requirements**

**3.1 Book Management**

* FR1: Librarian can add a new book (title, author, ISBN, quantity, genre).
* FR2: Librarian can update or delete existing book records.
* FR3: System should allow searching books by title, author, or ISBN.

**3.2 Member Management**

* FR4: Librarian can register new members (name, contact info, ID).
* FR5: Librarian can update/delete member information.
* FR6: System should search members by name or ID.

**3.3 Issue/Return Management**

* FR7: Librarian can issue a book to a member.
* FR8: System checks availability before issuing.
* FR9: Librarian can process the return of a book.
* FR10: System calculates and stores overdue fines based on return date.

**3.4 Reporting**

* FR11: Generate reports on issued/returned books
* FR12: Generate report of overdue books with fine amounts
* FR13: Summary report of library inventory

**4. Non-Functional Requirements**

**4.1 Performance Requirements**

* The system must support at least 10 concurrent users.
* Operations like book search should complete within 2 seconds.

**4.2 Reliability & Availability**

* The system should be available 95% of the time.
* Daily database backups should be supported.

**4.3 Security Requirements**

* Only authenticated users (librarians) can access administrative functions.
* Data should be protected using role-based access control.

**4.4 Usability**

* The UI must be simple and intuitive for non-technical users.
* Input validations should prevent incorrect data entry.

**4.5 Maintainability**

* The codebase should be modular and documented.
* Database schema should support future features like reservations.

**5. System Models (Optional)**

* ER Diagram (Entity-Relationship)
* Use Case Diagrams
* Sequence Diagrams

**6. Appendices**

* Sample Book and Member Form Screens
* Fine Calculation Rules
* Login/Logout Flow Description

**Software Analysis**

**Q.19. Perform a functional analysis for an online shopping system.**

**Ans.** Functional Analysis – Online Shopping System

1. **System Overview**

The Online Shopping System enables users to browse products, manage carts, make purchases, and track orders. The system must also support administrative functions for product and order management.

**2. Primary Actors (Users)**

|  |  |
| --- | --- |
| **Actor** | **Role** |
| Customer | Browses, buys products, tracks orders |
| Admin | Manages inventory, orders, customers |
| Delivery Agent *(optional)* | Updates delivery status |
| Guest User | Limited access to view products (without account) |

**3. Functional Requirements (FR)**

**3.1 User Account Management**

* FR1: Users can register and log in using email/password or social login.
* FR2: Users can manage profile details (address, phone number, etc.).
* FR3: Admin can create, delete, or update user accounts.

**3.2 Product Browsing and Search**

* FR4: Users can view product categories and individual products.
* FR5: Users can search and filter products by keyword, price, rating, etc.
* FR6: Each product has a description, price, image, availability, and rating.

**3.3 Shopping Cart and Wishlist**

* FR7: Users can add/remove products to/from the cart.
* FR8: Cart automatically updates subtotal and total amounts.
* FR9: Users can create and manage a wishlist.

**3.4 Order and Checkout Process**

* FR10: Users can place orders by selecting a delivery address and payment method.
* FR11: System should support multiple payment options (Credit Card, UPI, COD).
* FR12: Confirmation email/SMS is sent after successful payment.

**3.5 Order Tracking and History**

* FR13: Users can view order history and current status (e.g., shipped, delivered).
* FR14: Admin can update order status and notify the customer.

**3.6 Product Reviews and Ratings**

* FR15: Users can submit product ratings and reviews after purchase.
* FR16: Admin can moderate/delete inappropriate reviews.

**3.7 Inventory Management (Admin)**

* FR17: Admin can add/edit/delete product listings.
* FR18: Admin can manage product stock levels and restock alerts.

**3.8 Promotions and Discounts**

* FR19: Admin can create discount codes or apply seasonal discounts.
* FR20: System should validate discount codes at checkout.

**3.9 Security & Authentication**

* FR21: System must use secure password storage (e.g., hashing).
* FR22: Sessions should time out after a defined period of inactivity.
* FR23: Sensitive transactions (like checkout) must use HTTPS.

### **Use Case Summary**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Actor** | **Brief Description** |
| Register/Login | Customer | Sign up or access the system |
| Browse/Search Products | Guest/Customer | View product details or search by category |
| Add to Cart | Customer | Select products for purchase |
| Place Order | Customer | Complete checkout and payment |
| Manage Products | Admin | Add, edit, delete product listings |
| Manage Orders | Admin | Track and update order status |
| Submit Review | Customer | Rate and review a purchased product |
| View Order Status | Customer | See delivery progress of purchased items |

**5. System Interfaces**

* **User Interface (UI)**: Web and/or mobile interface for customers
* **Admin Interface**: Dashboard for order, product, and user management
* **Database**: Stores users, products, orders, payments
* **Payment Gateway API**: External service integration for processing payments

**6. Dependencies and Constraints**

* Internet access is required for full functionality.
* Integration with third-party services like payment gateways, email/SMS services.
* Compliance with data protection regulations (e.g., GDPR, PCI-DSS for payments).

**System Design**

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

**Ans.** Here's a Basic System Architecture for a Food Delivery App that outlines all key components, their roles, and interactions. This architecture balances simplicity with scalability, suitable for a startup or MVP (Minimum Viable Product).

**1. Architecture Overview**

**This architecture consists of three major layers:**

* Client Layer (Frontend)
* Application Layer (Backend APIs)
* Data Layer (Database + External Services)

**+--------------------------+**

**| Client Layer |**

**| (Mobile App / Web App) |**

**+--------------------------+**

**|**

**↓**

**+--------------------------+**

**| Application Layer |**

**| (Backend APIs + Services)|**

**+--------------------------+**

**|**

**↓**

**+--------------------------+**

**| Data Layer |**

**| (Database + Integrations)|**

**+--------------------------+**

**2. Client Layer**

**Users:**

* Customer App (iOS/Android/Web)
* Restaurant Dashboard (Web)
* Delivery Partner App (Mobile)

**Functions:**

* Browse restaurants/menus
* Place orders and make payments
* Track order and delivery status
* Receive push notifications

**3. Application Layer (Backend)**

This layer handles the business logic, request routing, and communication with data sources and third-party services.

**Key Microservices (or modules in a monolithic system):**

**Technologies (Example):**

* Backend Framework: Node.js / Django / Spring Boot
* API Protocol: REST or GraphQL
* Authentication: OAuth 2.0, JWT
* Payment Integration: Razorpay, Stripe, Paytm

**4. Data Layer**

**Databases:**

* Relational DB: PostgreSQL / MySQL (orders, users, restaurants)
* NoSQL: MongoDB / Redis (menu cache, location, session data)
* GeoDB: Spatial data storage for GPS-based tracking

**Storage:**

* Cloud Storage (e.g., AWS S3): Restaurant images, receipts

**5. External Integrations**

* **Payment Gateways**: Stripe, Razorpay, Paytm
* **SMS/Email API**: Twilio, SendGrid
* **Maps & Geolocation**: Google Maps API, OpenStreetMap
* **Push Notifications**: Firebase Cloud Messaging (FCM)
* **Analytics & Crash Reporting**: Firebase / Mixpanel / Sentry

**🛡️ 6. Security Considerations**

* Data encryption (HTTPS, TLS)
* Token-based authentication (JWT)
* Role-based access control
* Input validation and rate limiting

**🔄 7. Optional Enhancements (Future)**

* AI-based Delivery Time Estimation
* Recommendation System for restaurants/dishes
* Chat Support Module for real-time assistance
* Loyalty/Rewards System

**Software Testing**

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

**Ans.** Test Cases – Simple Calculator

**🔧 Assumptions:**

* Inputs are numeric (integers or floats)
* Calculator performs basic arithmetic: +, -, ×, ÷
* Division by zero is handled
* User interacts via UI or command-line input

**Edge Case Suggestions**

* Multiple operators in sequence (e.g., 5 + + 2)
* Decimal division resulting in long precision (e.g., 1 ÷ 3)
* Input with unexpected symbols or non-numeric values
* Testing operation precedence (parentheses usage)

**Maintenance**

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

**Ans.** Real-World Case: Critical Maintenance of the Facebook iOS App Outage (May 2019)

**Summary**

In May 2019, Facebook’s iOS application experienced a massive crash event globally, affecting millions of users. The issue was not isolated to Facebook alone — other major apps like Spotify, TikTok, Pinterest, and Tinder also crashed unexpectedly on launch for iOS users.

**Date of Incident**

May 6–7, 2019

**Problem Description**

A software update to the Facebook SDK (Software Development Kit) for iOS contained a breaking change.

This SDK is integrated into thousands of iOS apps for features like login, analytics, and sharing.

Once the SDK update went live, apps that used the SDK began crashing immediately upon launch — even if users were not using any Facebook features.

The issue stemmed from a remote configuration change in Facebook’s SDK that was not tested adequately.

**Root Cause Analysis**

Facebook uses remote commands (feature flags/configs) to enable or disable SDK behaviors without updating apps.

A new configuration caused a call to a non-existent object, leading to a null pointer crash.

Because this configuration was not deployed using a proper staging or canary release, it affected all users at once.

**Symptoms**

* Affected apps crashed on startup.
* Users could not open apps like Spotify, TikTok, Pinterest.
* iOS crash logs pointed to Facebook SDK as the source.

**Maintenance Actions Taken**

1. **Immediate rollback** of the problematic configuration by Facebook.
2. App developers issued **statements** to clarify that their apps weren’t the root cause.
3. **Temporary workarounds** were released by developers to disable the Facebook SDK.
4. Facebook issued a **patch** to the SDK and published updated documentation.

**Key Lessons Learned**

* Even a third-party SDK can cause **widespread production outages**.
* Remote configurations should be **staged, tested, and rolled out gradually**.
* Critical dependencies must be isolated or **wrapped with fail-safes**.
* Developers should use **crash analytics tools** (e.g., Firebase Crashlytics, Sentry) to detect issues early.

**DFD (Data Flow Diagram)**

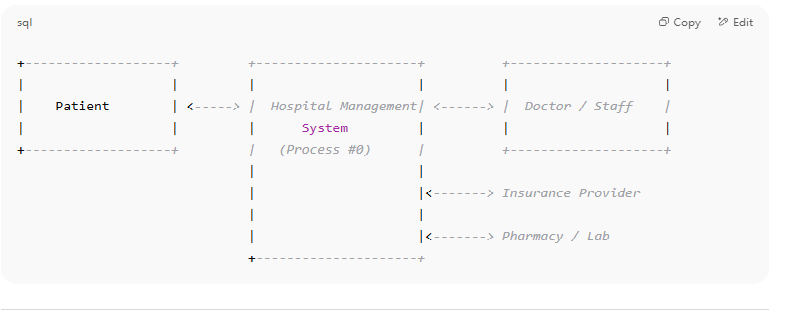
**Q.23. Create a DFD for a hospital management system.**

**Ans.** Here’s a Data Flow Diagram (DFD) for a Hospital Management System (HMS) — both Context Level (Level 0) and Level 1 diagrams are described.

**DFD for Hospital Management System**

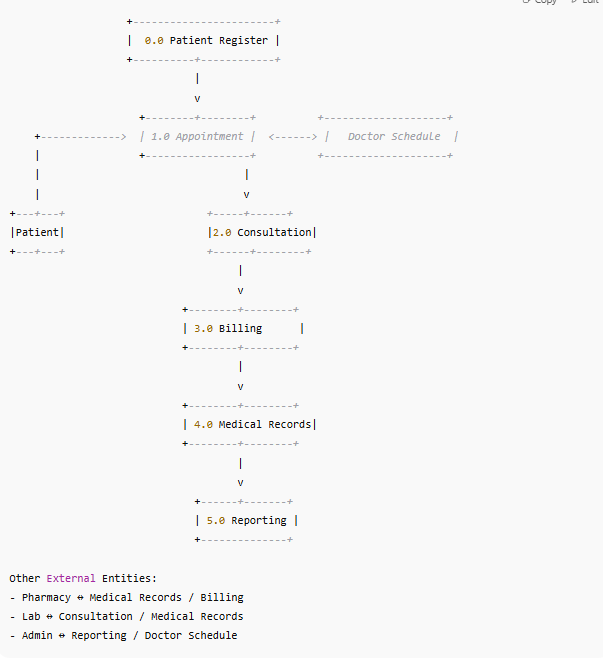
**1. Context Level DFD (Level 0)**

Purpose: Shows the entire HMS as a single process and its interaction with external entities.

****

**2. Level 1 DFD**

Breaks down Process #0 into multiple sub-processes:

****

**Data Stores (used in Level 1 DFD):**

* D1: Patient Database
* D2: Appointment Schedule
* D3: Doctor Records
* D4: Billing Information
* D5: Medical Records

**Desktop Application**

**Q.23. Build a simple desktop calculator application using a GUI library.**

**Ans.**

Simple Calculator (HTML + CSS + JS)

**Features:**

* Supports +, -, ×, ÷, decimals, and C (clear)
* Keyboard-style layout
* Responsive and clean

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Simple Calculator</title>

<style>

body {

background: #f0f0f0;

font-family: Arial, sans-serif;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.calculator {

background: #fff;

padding: 20px;

border-radius: 15px;

box-shadow: 0 5px 15px rgba(0,0,0,0.2);

width: 300px;

}

.display {

width: 100%;

height: 50px;

font-size: 24px;

text-align: right;

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 8px;

}

.buttons {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 10px;

}

.buttons button {

font-size: 20px;

padding: 15px;

border: none;

background: #e0e0e0;

border-radius: 8px;

cursor: pointer;

transition: 0.2s;

}

.buttons button:hover {

background: #d0d0d0;

}

.buttons .equal {

grid-column: span 2;

background: #4CAF50;

color: white;

}

.buttons .equal:hover {

background: #45a049;

}

.buttons .clear {

background: #f44336;

color: white;

}

.buttons .clear:hover {

background: #e53935;

}

</style>

</head>

<body>

<div class="calculator">

<input type="text" class="display" id="display" disabled>

<div class="buttons">

<button onclick="append('7')">7</button>

<button onclick="append('8')">8</button>

<button onclick="append('9')">9</button>

<button onclick="append('/')">÷</button>

<button onclick="append('4')">4</button>

<button onclick="append('5')">5</button>

<button onclick="append('6')">6</button>

<button onclick="append('\*')">×</button>

<button onclick="append('1')">1</button>

<button onclick="append('2')">2</button>

<button onclick="append('3')">3</button>

<button onclick="append('-')">−</button>

<button onclick="append('0')">0</button>

<button onclick="append('.')">.</button>

<button class="equal" onclick="calculate()">=</button>

<button onclick="append('+')">+</button>

<button class="clear" onclick="clearDisplay()">C</button>

</div>

</div>

<script>

const display = document.getElementById('display');

function append(value) {

display.value += value;

}

function calculate() {

try {

display.value = eval(display.value);

} catch {

alert("Invalid Expression");

clearDisplay();

}

}

function clearDisplay() {

display.value = '';

}

</script>

</body>

</html>

**How to Run:**

1. Copy the code above.
2. Paste it into a file and save as calculator.html.
3. Double-click the file to open it in your browser.

**Flow Chart**

**Q.24. Draw a flowchart representing the logic of a basic online registration system.**

**Ans.** Here is a simple way to draw a flowchart in HTML using pure HTML and CSS with a bit of layout styling. This represents the logic of a basic online registration system, including decision steps like form validation.

**Flowchart: Basic Online Registration System**

This version uses **HTML and CSS** only (no external libraries like Flowchart.js or SVG).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Online Registration Flowchart</title>

<style>

body {

font-family: Arial, sans-serif;

background: #f9f9f9;

padding: 40px;

display: flex;

justify-content: center;

}

.flowchart {

display: flex;

flex-direction: column;

align-items: center;

}

.box {

background: #fff;

padding: 15px 25px;

border: 2px solid #333;

border-radius: 10px;

margin: 10px;

text-align: center;

width: 220px;

position: relative;

}

.arrow {

width: 2px;

height: 30px;

background: #333;

}

.decision {

border-radius: 0;

transform: rotate(45deg);

width: 100px;

height: 100px;

display: flex;

align-items: center;

justify-content: center;

}

.decision span {

transform: rotate(-45deg);

}

.yesno {

display: flex;

justify-content: space-between;

width: 250px;

}

.yesno .arrow {

height: 2px;

width: 100px;

background: #333;

margin: 10px;

}

.yes-label, .no-label {

font-size: 14px;

margin-top: -15px;

}

</style>

</head>

<body>

<div class="flowchart">

<div class="box">Start</div>

<div class="arrow"></div>

<div class="box">Fill Registration Form</div>

<div class="arrow"></div>

<div class="box decision"><span>Is Form Valid?</span></div>

<div class="yesno">

<div class="yes-label">Yes</div>

<div class="no-label">No</div>

</div>

<div class="yesno">

<div class="arrow" style="transform: rotate(-45deg);"></div>

<div class="arrow" style="transform: rotate(45deg);"></div>

</div>

<div class="yesno">

<div class="box" style="margin-left: -50px;">Show Error Message</div>

<div class="box" style="margin-right: -50px;">Save to Database</div>

</div>

<div class="arrow" style="margin-top: -20px;"></div>

<div class="box">Send Confirmation Email</div>

<div class="arrow"></div>

<div class="box">End</div>

</div>

</body>

</html>

**Steps Represented in the Flowchart:**

1. **Start**
2. **Fill Registration Form**
3. **Is Form Valid?**
   * If **No** → Show Error Message
   * If **Yes** → Save to Database → Send Email
4. **End**

**How to Use:**

1. Save the code as registration-flowchart.html
2. Open it in any modern web browser