**Module 1 – Overview of IT Industry**

**What is a Program?**

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

**Hello World in C:**

#include <stdio.h> // Preprocessor directive

int main() { // Main function - entry point of the program

printf("Hello, World!\n"); // Output statement

return 0; // Return value to the OS

}

**Hello World in C++:**

#include <iostream> // Preprocessor directive

int main() { // Main function - entry point of the program

std::cout << "Hello, World!" << std::endl; // Output statement

return 0; // Return value to the OS

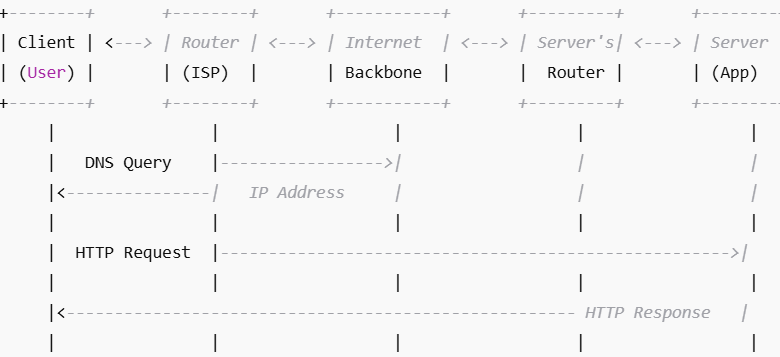
}

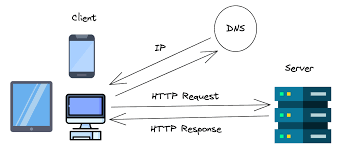
| **Feature** | **C** | **C++** |
| --- | --- | --- |
| Based on | Procedural programming | Object-Oriented + Procedural |
| Output method | printf() with format specifiers | cout with stream insertion << |
| Header | <stdio.h> | <iostream> |
| Namespaces | Not supported | Supported (std::) |
| Flexibility | Low-level, simple | More features and flexibility |
| Output buffering | \n just adds newline | std::endl adds newline and flushes buffer |

**---------------------------------------------------------------------------------------------------------**

**World Wide Web & How Internet Works**

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

****



**---------------------------------------------------------------------------------------------------------**

**Network Layers on Client and Server**

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

**Simple HTTP Server (Python)**

from http.server import BaseHTTPRequestHandler, HTTPServer

class MyRequestHandler(BaseHTTPRequestHandler):

def do\_GET(self):

# Respond with a simple message

self.send\_response(200)

self.send\_header('Content-type', 'text/html')

self.end\_headers()

message = "<h1>Hello from the Server!</h1>"

self.wfile.write(message.encode('utf-8'))

# Define host and port

host = 'localhost'

port = 8080

# Start the server

server = HTTPServer((host, port), MyRequestHandler)

print(f"Server started at http://{host}:{port}")

server.serve\_forever()

**Simple HTTP Client (Python)**

import requests

# Send GET request to the server

response = requests.get("http://localhost:8080")

# Print server's response

print("Status Code:", response.status\_code)

print("Response Body:", response.text)

**--------------------------------------------------------------------------------------------------------------------------**

**Types of Internet Connections**

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

| **Type of Connection** | **Description** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Fiber-Optic** | Uses light signals through thin glass or plastic fibers to transmit data. Very high-speed and modern. | - Extremely fast (up to 1 Gbps or more) - Low latency - Highly reliable - Ideal for streaming, gaming, cloud | - Expensive infrastructure - Limited availability in rural areas - Installation can be complex |
| **Cable Broadband** | Uses coaxial cables (same as cable TV) to deliver high-speed internet. | - Fast speeds (100–1000 Mbps) - Widely available in urban areas - Easy to bundle with TV services | - Speed may drop during peak hours (shared bandwidth) - Slightly higher latency than fiber |
| **DSL (Digital Subscriber Line)** | Transmits data over standard telephone lines without interrupting phone use. | - Inexpensive - Widely available - Dedicated line (not shared) | - Slower speeds (5–50 Mbps) - Speed decreases with distance from ISP - Outdated in many areas |
| **Satellite Internet** | Uses communication satellites in orbit to beam internet signals to a satellite dish at the user’s location. | - Available almost anywhere (great for rural or remote areas) - Independent of land infrastructure | - High latency (due to long distance to satellite) - Data caps and expensive - Weather can disrupt signal |
| **Mobile/Cellular (4G/5G)** | Provides internet through mobile networks, using SIM-enabled devices or hotspots. | - Portable (can use on-the-go) - 5G offers high speeds and low latency - No cables needed | - Data limits/costly plans - Speed depends on signal strength - 5G availability still expanding |
| **Fixed Wireless** | Transmits data wirelessly between two fixed points (e.g., ISP tower to rooftop antenna). | - Good for rural areas - No cables required - Lower latency than satellite | - Requires line-of-sight to tower - Affected by weather - Speed varies (10–100 Mbps) |
| **Dial-Up** | Connects via traditional telephone line, using a modem. Very outdated. | - Very cheap - Available anywhere with a phone line | - Very slow (max 56 Kbps) - Ties up phone line - Incompatible with modern internet usage |

**--------------------------------------------------------------------------------------------------------------------------**

**Protocols**

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

**What is curl?**

* **curl stands for Client URL.**
* **It is used to make network requests from the command line.**
* **Supports many protocols: HTTP, HTTPS, FTP, FTPS, SCP, SFTP, SMTP, etc.**

**Part 1: Simulate HTTP Requests using curl**

HTTP (HyperText Transfer Protocol) is the foundation of data communication for the web.

**📥 1. HTTP GET Request (Fetching a web page)**

bash

code

curl http://example.com

🔍 Explanation:

* This sends an HTTP GET request to http://example.com.
* Returns the HTML content of the page.

**📄 2. Save the Output to a File**

bash

code

curl http://example.com -o output.html

🔍 Saves the response into a file called output.html.

**📡 3. View HTTP Headers**

bash

code

curl -I http://example.com

🔍 -I or --head fetches only the **HTTP headers**, not the body.

Example Output:

yaml

code

HTTP/1.1 200 OK

Content-Type: text/html; charset=UTF-8

Date: Sun, 13 Jul 2025 05:30:00 GMT

**✅ 4. Send a POST Request with Data**

bash

code

curl -X POST -d "username=dev&password=1234" http://example.com/login

🔍 Explanation:

* -X POST specifies the POST method.
* -d sends data as application/x-www-form-urlencoded.

**🔐 5. Send Request Over HTTPS**

bash

code

curl https://example.com

🔍 Secure version of HTTP. Data is encrypted using SSL/TLS.

**🧪 6. Set Custom Headers**

bash

code

curl -H "User-Agent: MyTestAgent" http://example.com

🔍 Sends a custom header (User-Agent) to simulate a specific client.

**📂 7. Download a File**

bash

code

curl -O https://example.com/file.zip

🔍 -O saves the file with the original filename (file.zip in this case).

**Part 2: Simulate FTP Requests using curl**

FTP (File Transfer Protocol) is used to upload/download files from an FTP server.

**🔐 1. Connect and Download a File from FTP**

bash

code

curl -u username:password ftp://ftp.example.com/myfile.txt -o myfile.txt

🔍 Explanation:

* -u provides login credentials.
* Downloads myfile.txt from the FTP server.

**📤 2. Upload a File to an FTP Server**

bash

code

curl -T upload.txt -u username:password ftp://ftp.example.com/uploads/

🔍 Explanation:

* -T specifies the file to upload.
* It uploads upload.txt to the /uploads/ directory.

**📜 3. List Files in an FTP Directory**

bash

code

curl -u username:password ftp://ftp.example.com/

🔍 Lists the contents of the root directory.

**🔐 4. Use FTPS (FTP Secure over SSL/TLS)**

bash

code

curl --ftp-ssl -u username:password ftp://ftp.example.com/myfile.txt

🔍 FTPS encrypts FTP communication for security.

**--------------------------------------------------------------------------------------------------------------------------**

**Application Security**

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

**✅ Common Application Security Vulnerabilities**

| **Vulnerability** | **Explanation** | **Possible Solutions** |
| --- | --- | --- |
| **SQL Injection** | Malicious SQL code is inserted into a query, allowing attackers to access or modify the database. | - Use **prepared statements** or **ORMs** - **Input validation & sanitization** |
| **Cross-Site Scripting (XSS)** | Attacker injects malicious scripts into web pages viewed by other users. | - **Escape HTML** output - Use **Content Security Policy (CSP)** - Validate inputs |
| **Cross-Site Request Forgery (CSRF)** | Tricks a user into unknowingly submitting malicious requests from a trusted session. | - Use **anti-CSRF tokens** - **SameSite** cookie attribute - User re-authentication |

✅ **SQL Injection** – attack on the database → fix with **prepared statements**.

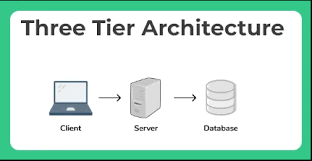
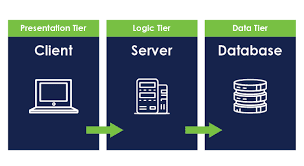
✅ **XSS** – attack on users’ browsers → fix with **escaping** and **CSP**.

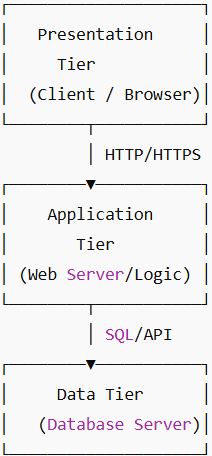
✅ **CSRF** – abuse of user’s session → fix with **tokens** and **SameSite cookies**.

**--------------------------------------------------------------------------------------------------------------------------**

**Software Architecture**

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



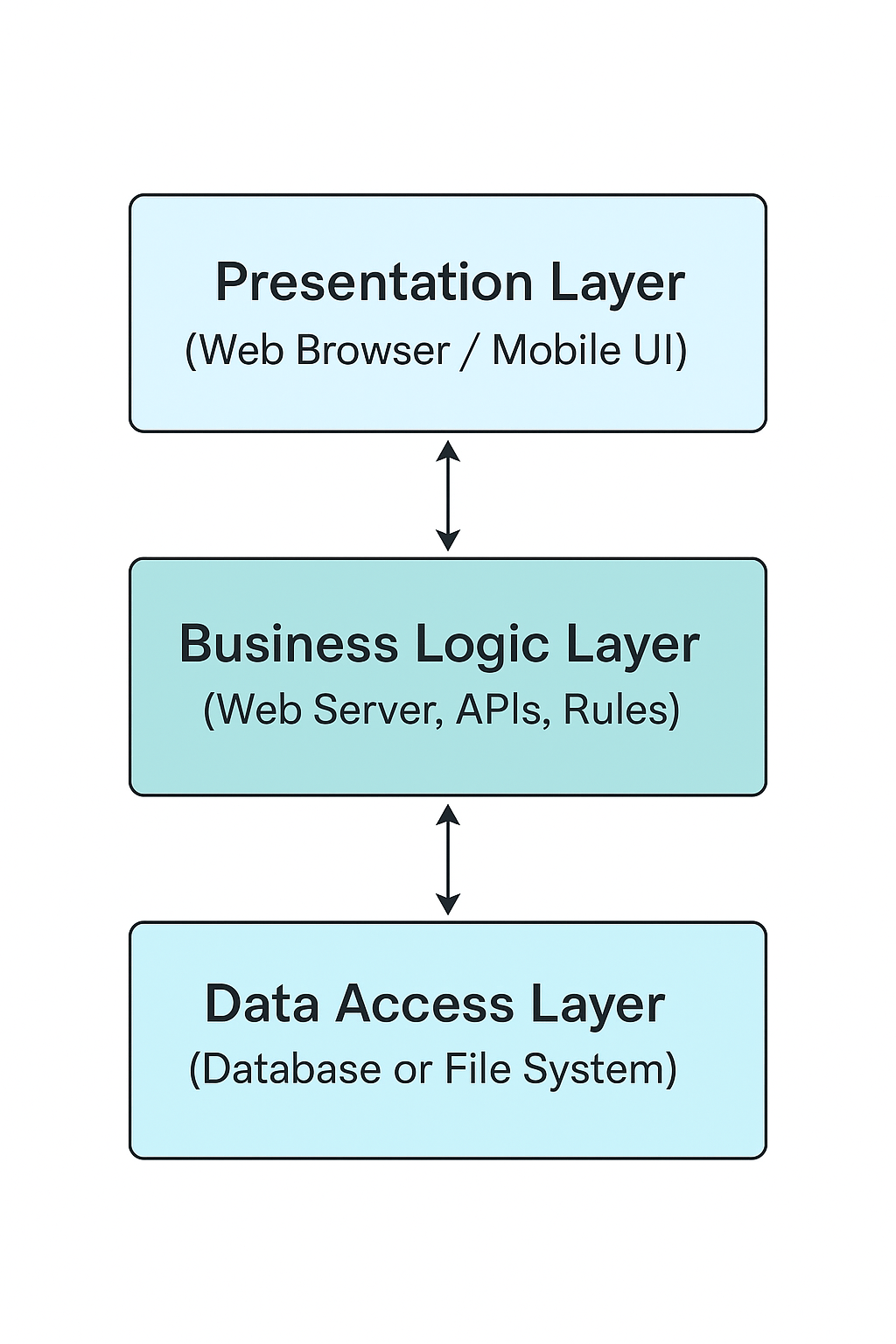


| **Tier** | **Role** | **Examples** |
| --- | --- | --- |
| **1. Presentation** | User interface layer; displays data and captures input | Web browser, HTML/CSS/JS frontend |
| **2. Application** | Processes business logic, handles requests and responses | Web server, APIs, backend logic (Node.js, Java, Python) |
| **3. Data** | Stores and retrieves data as requested by application tier | SQL/NoSQL database servers (MySQL, MongoDB) |

**--------------------------------------------------------------------------------------------------------------------------**

**Layers in Software Architecture**

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



**🛒 Case Study: Online Shopping Web Application**

This case study examines a typical **online shopping web application** (like Amazon or Flipkart) using a **three-tier architecture**. The architecture separates the system into three layers: **Presentation Layer**, **Business Logic Layer**, and **Data Access Layer**, each responsible for distinct functionality.

**1️⃣ Presentation Layer**

* **Role**: Acts as the user interface (UI), allowing users to interact with the application.
* **Examples**: Web browser, mobile app.
* **Functions**:
  + Displays products
  + Accepts user input (login, search, cart actions)
  + Shows confirmations and error messages

**2️⃣ Business Logic Layer**

* **Role**: Processes all core functionalities and business rules.
* **Examples**: Web server, backend API.
* **Functions**:
  + Authenticates users
  + Manages product searches and filters
  + Validates orders, applies discounts
  + Controls the checkout and payment workflow

**3️⃣ Data Access Layer**

* **Role**: Handles storage and retrieval of data from the database.
* **Examples**: SQL/NoSQL database systems.
* **Functions**:
  + Retrieves product details
  + Stores user and order information
  + Updates inventory
  + Saves transaction and payment records

**--------------------------------------------------------------------------------------------------------------------------**

**Software Environments**

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

**Types of Software Environments**

| **Environment** | **Purpose** | **Characteristics** |
| --- | --- | --- |
| **Development** | For writing and building code | - Used by developers - Debugging tools available - Frequent code changes |
| **Testing** | For testing new features and bug fixes | - Runs test cases - Simulates production environment - Often includes CI/CD |
| **Staging** (optional) | A mirror of production for final testing | - Pre-production environment - Tests deployment and performance |
| **Production** | Live environment accessed by end users | - High performance & security - Real user data - No direct changes allowed |

**Set Up a Basic Environment in a Virtual Machine (Linux-Based)**

Let’s create a **basic development environment** in a **VM** using **Ubuntu**.

**📦 Requirements:**

* VirtualBox (or VMware)
* Ubuntu ISO (from [ubuntu.com](https://ubuntu.com/download))
* 8 GB disk, 2 GB RAM recommended

| **Step** | **Description** |
| --- | --- |
| 1. **Install VM** | Use VirtualBox to create a new VM and install Ubuntu from ISO. |
| 2. **Update System** | sudo apt update && sudo apt upgrade |
| 3. **Install Dev Tools** | sudo apt install git curl build-essential |
| 4. **Install Language Stack** | Example: sudo apt install python3 python3-pip or sudo apt install openjdk-17-jdk |
| 5. **Install Code Editor** | Example: VS Code – sudo snap install code --classic |
| 6. **Create a Sample Project** | mkdir my-app && cd my-app && code . |
| 7. **(Optional) Set Up Web Server** | sudo apt install apache2 or npm install -g http-server |

**--------------------------------------------------------------------------------------------------------------------------**

**Source Code**

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

Simple source code:

#include <stdio.h>

int main() {

printf("Hello, GitHub!\n");

return 0;

}

| **Step** | **Action** |
| --- | --- |
| **1. Create a GitHub Account** | Go to [https://github.com](https://github.com/) and sign up if you don’t already have an account. |
| **2. Create a New Repository** | Click + → **New repository** → Name it (e.g., hello-world) → Add description (optional) → Click **Create repository**. |
| **3. Write a Source Code File** | Open a code editor (e.g., VS Code or Notepad). Write a simple file, e.g., hello.c: print("Hello, GitHub!") |
| **4. Save the File** | Save the file locally on your computer (e.g., hello.c). |
| **5. Initialize Git (Optional CLI Method)** | Open terminal or command prompt:bash<br>git init<br>git add hello.py<br>git commit -m "First commit"<br>git remote add origin https://github.com/yourusername/hello.git<br>git push -u origin master<br> |
| **6. Or Upload via Web Interface** | Go to your repository → Click **Add file** → **Upload files** → Select hello.c → Click **Commit changes**. |

**--------------------------------------------------------------------------------------------------------------------------**

**Github and Introductions**

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

**✅ Part 1: Create a GitHub Repository**

1. Go to [https://github.com](https://github.com/)
2. Click the **“+”** icon (top right) → **“New repository”**
3. Fill in:
   * **Repository name**: my-first-repo
   * **Description**: (optional)
   * Choose **Public** or **Private**
   * Optionally, check **"Add a README"**
4. Click **“Create repository”**

**✅ Part 2: Commit and Push Code Using Git CLI**

**🖥️ Prerequisites:**

* Git must be installed → [https://git-scm.com](https://git-scm.com/)
* GitHub account must be created
* Terminal or Command Prompt access

**🔧 Step-by-Step Commands:**

**📝 1. Write Your Code**

Create a file hello.py:

print("Hello, GitHub!")

**🔁 2. Initialize Git Repository**

Open terminal in your project folder:

git init

**➕ 3. Add Files to Staging**

git add hello.py

**✅ 4. Commit Changes**

git commit -m "Initial commit: Add hello.py"

**🌐 5. Link to GitHub Remote Repository**

Replace your-username with your GitHub username:

git remote add origin https://github.com/your-username/my-first-repo.git

**⬆️ 6. Push Code to GitHub**

git push -u origin master

📝 If main is the default branch on GitHub, use:

git push -u origin main

**--------------------------------------------------------------------------------------------------------------------------**

**Student Account in Github**

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

**Part 1: Create a GitHub Student Account**

1. 🔗 Go to: <https://education.github.com/pack>
2. Click **“Get Student Benefits”**
3. Sign in with your **GitHub account**
4. Fill in:
   * School name
   * Student email or upload your student ID
   * Expected graduation year
5. Submit your application
6. ✅ Wait for approval (usually 1–2 days)

**Part 2: Collaborate on a GitHub Project with a Classmate**

**Step-by-Step Collaboration:**

| **Step** | **Action** |
| --- | --- |
| **1. Create Repository** | One student creates a new GitHub repository (e.g., student-project). |
| **2. Add Collaborator** | Go to **Settings → Collaborators** → Enter your classmate’s GitHub username → Click **Add**. |
| **3. Clone the Repo** | Both students run: git clone https://github.com/username/student-project.git |
| **4. Make Changes** | Each student edits files, adds features or fixes. |
| **5. Commit Changes** | Use: git add . git commit -m "Your message" |
| **6. Push Changes** | Use: git push origin main |
| **7. Pull Updates** | Regularly run: git pull origin main to sync the latest changes. |

**--------------------------------------------------------------------------------------------------------------------------**

**Types of Software**

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

| **Category** | **Examples** | **Description** |
| --- | --- | --- |
| **System Software** | 1. Windows 10/11  2. Linux (Ubuntu)  3. macOS  4. Android OS  5. Device Drivers | Core software that manages hardware and system resources |
| **Application Software** | 1. MS Word  2. Google Chrome  3. Zoom  4. Spotify  5. Adobe Photoshop | Programs that help users perform specific tasks |
| **Utility Software** | 1. WinRAR / 7-Zip  2. Antivirus (e.g., Avast)  3. Disk Cleanup Tool  4. CCleaner  5. Backup Software | Tools to maintain, optimize, or protect the computer system |

**System Software**: Runs the system (OS, drivers)

**Application Software**: Helps you do work/play (apps like Word, Zoom)

**Utility Software**: Keeps your system clean, secure, and efficient

| **Category** | **Function** | **Used By** | **Examples** |
| --- | --- | --- | --- |
| System Software | Manages hardware and runs the system | OS and developers | Windows, Linux, Android |
| Application Software | Helps users perform specific tasks | General users | MS Word, Chrome, Zoom |
| Utility Software | Optimizes or maintains the system | Users & administrators | Antivirus, WinRAR, CCleaner |

**--------------------------------------------------------------------------------------------------------------------------**

**GIT and GITHUB Training**

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

Sure! Here's a **brief step-by-step Git tutorial** to help you **practice cloning, branching, and merging** repositories.

**✅ Step-by-Step Git Tutorial**

**🧰 Pre-requisites:**

* Git installed on your system ([Download Git](https://git-scm.com/downloads))
* A GitHub account

**🔁 Step 1: Clone a Repository**

1. Go to a GitHub repository (e.g., your own or a class project).
2. the repo URL (HTTPS).
3. Open terminal/command prompt and run:

git clone https://github.com/your-username/repo-name.git

cd repo-name

✅ *You now have a local of the project.*

**🌿 Step 2: Create a New Branch**

git checkout -b feature-branch

* This creates and switches to a new branch called feature-branch.

✏️ Make some changes (edit a file or create a new one).

**💾 Step 3: Commit Your Changes**

git add .

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

* git add . stages all changes
* git commit saves them with a message

**🔀 Step 4: Switch Back to Main Branch**

git checkout main

**🔄 Step 5: Merge the Feature Branch into Main**

git merge feature-branch

* This applies changes from feature-branch into main

✅ *If there’s a conflict, Git will prompt you to resolve it.*

**⬆️ Step 6: Push Changes to GitHub**

git push origin main

* This updates the GitHub repository with your merged changes.

**🧹 Optional: Delete the Feature Branch**

git branch -d feature-branch

**--------------------------------------------------------------------------------------------------------------------------**

**Application Software**

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

**🧾 Introduction**

Application software refers to programs designed to help users perform specific tasks on a computer or mobile device. Unlike system software, which runs the machine, **application software focuses on solving user problems**, increasing efficiency, and enhancing productivity across personal, academic, and professional settings.

**💡 Types of Application Software and Their Productivity Benefits**

| **Type** | **Description** | **Examples** | **Productivity Impact** |
| --- | --- | --- | --- |
| **Word Processing Software** | Used to create, edit, and format text documents. | Microsoft Word, Google Docs | Speeds up document creation, formatting, and collaboration. |
| **Spreadsheet Software** | Used for organizing data, performing calculations, and generating charts. | Microsoft Excel, Google Sheets | Automates calculations and data analysis, improving decision-making. |
| **Presentation Software** | Helps in creating slideshows for meetings, lectures, or reports. | Microsoft PowerPoint, Google Slides | Improves communication and clarity of ideas visually. |
| **Database Software** | Stores, manages, and retrieves structured data efficiently. | Microsoft Access, MySQL, Oracle | Organizes large data sets and enhances data accessibility and integrity. |
| **Email & Communication Tools** | Enables sending messages, video calls, and collaboration in real-time. | Gmail, Outlook, Zoom, Microsoft Teams | Saves time, supports remote work, and improves teamwork. |
| **Graphic Design Software** | Used to create visual content such as logos, images, and illustrations. | Adobe Photoshop, Canva | Enhances creativity and brand communication with visual tools. |
| **Web Browsers** | Allows users to access and interact with websites and online services. | Google Chrome, Mozilla Firefox | Enables access to online resources, tools, and cloud platforms. |
| **Project Management Software** | Helps plan, track, and manage projects and tasks. | Trello, Asana, Microsoft Project | Increases team coordination, task tracking, and time management. |
| **Accounting Software** | Manages financial data, invoices, taxes, and payroll. | Tally, QuickBooks, Zoho Books | Automates financial tasks and ensures accuracy in bookkeeping. |
| **Educational Software** | Supports learning through tutorials, quizzes, and interactive modules. | Duolingo, Khan Academy, Coursera | Enhances learning flexibility and access to quality resources. |

**📈 How Application Software Improves Productivity**

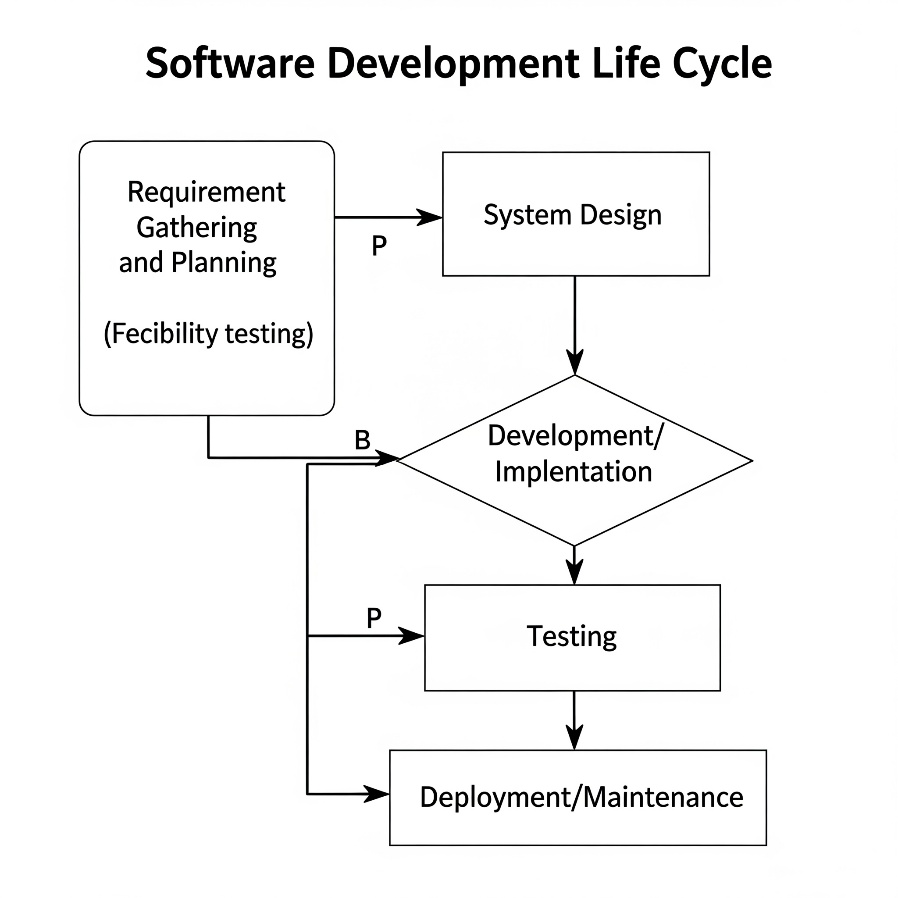
* 🕒 **Time-Saving**: Automates repetitive tasks (e.g., calculations, formatting, communication).
* 📊 **Data Management**: Organizes and analyzes data quickly (e.g., spreadsheets, databases).
* 👥 **Collaboration**: Allows real-time editing and teamwork (e.g., Google Docs, Teams).
* 🎨 **Creative Support**: Enhances design, writing, and presentation skills (e.g., Canva, Word).
* 📚 **Learning & Skill Development**: Makes learning accessible and engaging (e.g., Coursera).

**✅ Conclusion**Application software plays a crucial role in **enhancing user productivity** by enabling efficient task completion, facilitating communication, and streamlining workflows. In both academic and professional environments, using the right application tools can save time, reduce errors, and improve output quality significantly.

**--------------------------------------------------------------------------------------------------------------------------**

**Software Development Process**

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

****

**P (Proceed),B (Back/Return):**

This flowchart illustrates the typical stages involved in developing software:

* **Requirement Gathering and Planning**: This initial phase involves understanding the needs of the users and the system. It also includes **feasibility testing** to determine if the project is viable from technical, economic, and operational perspectives.
* **System Design**: In this stage, the architecture of the software is designed, including modules, databases, and user interfaces.
* **Development/Implementation**: This is where the actual coding takes place, translating the design into functional software.
* **Testing**: The developed software undergoes rigorous testing to identify and fix bugs, ensuring it meets the specified requirements and quality standards.
* **Deployment/Maintenance**: Once tested, the software is deployed to the production environment. This phase also includes ongoing maintenance, updates, and support to ensure the system continues to function effectively.

**--------------------------------------------------------------------------------------------------------------------------**

**Software Requirement**

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

**📘 Software Requirement Specification (SRS): Library Management System**

**✅ 1. Introduction**

**Purpose**

The purpose of this software is to design a **Library Management System (LMS)** that automates common library operations such as **adding/removing books, issuing books, returning books**, and **tracking due dates** for users like students and librarians.

**Scope**

This system will:

* Allow librarians to manage book inventory
* Allow users to search and borrow books
* Track issued books and due dates
* Generate basic reports on borrowed/returned books

**✅ 2. Overall Description**

**Users of the System**

* **Librarian (Admin)**: Full control over the system (add/delete books, manage users)
* **Students/Users**: Can search, borrow, and return books

**Functional Environment**

* Web-based or Desktop-based interface
* Backend database (e.g., MySQL)
* Developed using Python/Java/PHP with HTML/CSS/JS (if web)

**✅ 3. Functional Requirements**

| **ID** | **Requirement Description** |
| --- | --- |
| FR1 | System shall allow the admin to add/edit/delete books. |
| FR2 | Users shall be able to search books by title, author, or ID. |
| FR3 | Admin shall issue books to users and update the due date. |
| FR4 | Users shall return books and system will calculate any fines. |
| FR5 | Admin shall view a report of issued, returned, or overdue books. |
| FR6 | Login system shall be implemented for user authentication. |

**4. Non-Functional Requirements**

| **ID** | **Requirement Description** |
| --- | --- |
| NFR1 | The system should be responsive and load within 3 seconds. |
| NFR2 | The system should be secure and protect user data. |
| NFR3 | The interface should be user-friendly and intuitive. |
| NFR4 | System should be compatible with Windows, Linux (if desktop), or modern browsers (if web). |

**5. System Features**

| **Feature** | **Description** |
| --- | --- |
| Book Management | Add, edit, delete book records. |
| User Registration | Register and authenticate users. |
| Book Search | Search books by various criteria. |
| Issue/Return Books | Track borrow and return actions. |
| Fine Calculation | Automatically calculate overdue charges. |

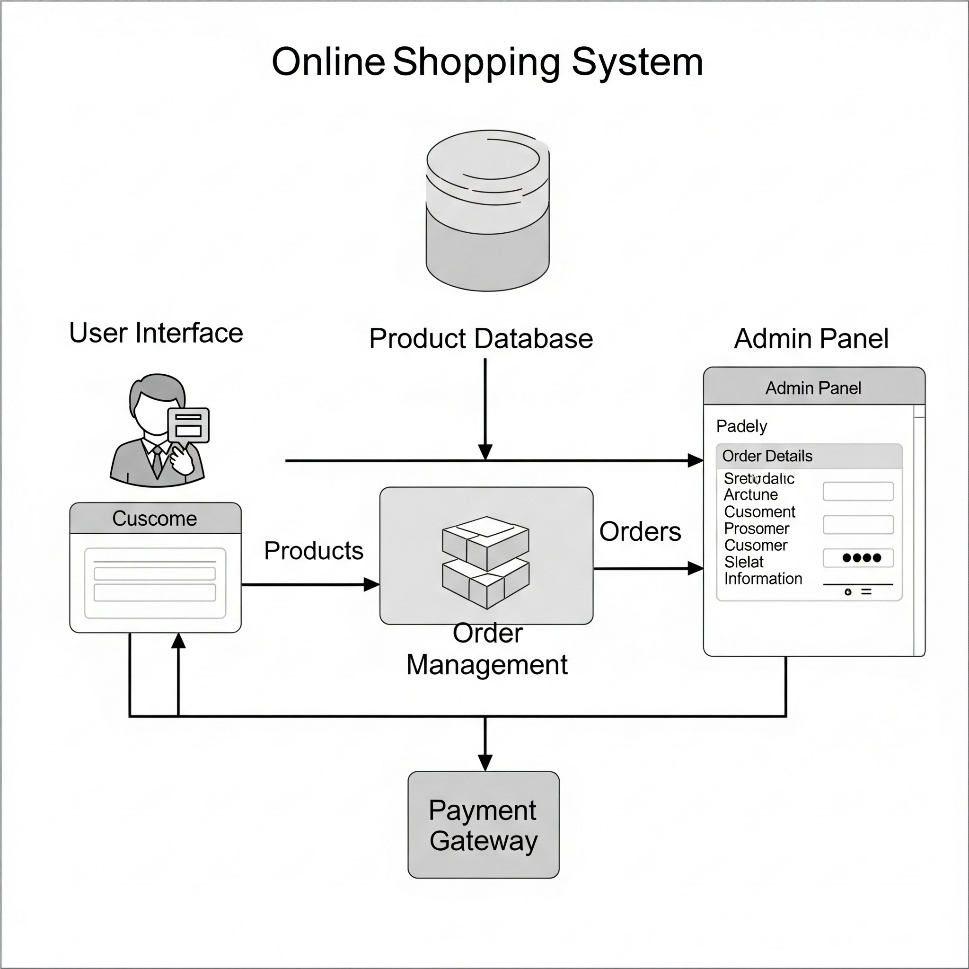
**6. Assumptions and Constraints**

* Each user can borrow a maximum of 3 books at a time.
* Book return period is 14 days.
* Fine is ₹5 per day after due date.
* Database should back up daily.

**-----------------------------------------------------------------------------------------------------------------**

**Software Analysis**

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

****

**Functional Analysis: Online Shopping System**

A functional analysis aims to identify and describe the essential operations or **functions** that a system must perform to meet user needs. For an online shopping system, these functions revolve around enabling users to browse products, make purchases, and manage their orders, while also allowing administrators to manage the system.

| **Actor** | **Role in the System** |
| --- | --- |
| **Customer** | Browses products, places orders, makes payments |
| **Admin** | Manages inventory, orders, users |
| **Delivery Staff** | Updates delivery status |
| **Payment Gateway** | Handles secure online transactions |

**Core Functional Areas and Their Features**

Here's a breakdown of the main functional areas, detailing the specific functions within each:

**1. User Management**

This area handles everything related to user accounts and profiles.

* Registration: Allows new users to create an account by providing necessary details (e.g., name, email, password).
* Login/Logout: Enables registered users to securely access and exit their accounts.
* Profile Management: Allows users to view and update their personal information, shipping addresses, and payment methods.
* Password Reset: Provides a mechanism for users to reset forgotten passwords.
* Order History View: Allows registered users to view their past orders.

**2. Product Management (Admin Functions)**

These functions are primarily for administrators to manage the product catalog.

* Add Product: Allows administrators to add new products to the system with details such as name, description, price, stock, images, and category.
* Edit Product: Enables administrators to modify existing product information.
* Delete Product: Allows administrators to remove products from the catalog.
* Manage Product Categories: Create, edit, and delete categories for product organization.
* Manage Product Attributes: Define attributes like size, color, or material for products.

**3. Product Browse & Search**

This covers how users discover products.

* Browse Products by Category: Allows users to navigate products through a hierarchical category structure.
* Search Products: Enables users to find products using keywords, with options for filtering and sorting (e.g., by price, relevance, brand).
* View Product Details: Displays comprehensive information about a selected product, including images, description, price, availability, and customer reviews.
* View Related Products: Suggests other products based on Browse history or product similarity.

**4. Shopping Cart Management**

This handles the process of selecting items for purchase.

* Add to Cart: Allows users to add desired products to a virtual shopping cart.
* View Cart: Displays the contents of the shopping cart, including product details, quantities, and subtotal.
* Update Cart Quantity: Enables users to change the quantity of items in the cart.
* Remove from Cart: Allows users to delete items from the cart.
* Apply Coupon/Discount Code: Enables users to apply promotional codes to their order.

**5. Checkout & Order Management**

This is the core purchasing functionality.

* Initiate Checkout: Starts the purchasing process from the shopping cart.
* Select Shipping Address: Allows users to choose or add a shipping address.
* Select Payment Method: Enables users to choose from available payment options (e.g., credit card, debit card, net banking).
* Place Order: Confirms the order and initiates the payment process.
* Order Confirmation: Provides users with a confirmation of their order, including an order ID and summary.
* Track Order Status: Allows customers to view the current status of their placed orders (e.g., pending, processing, shipped, delivered).
* Cancel Order: Allows customers (within a defined timeframe) or administrators to cancel an order.

**6. Payment Processing**

This crucial function integrates with payment gateways.

* Process Payment: Integrates with external payment gateways to securely process transactions.
* Handle Payment Success/Failure: Manages the outcomes of payment attempts and provides appropriate feedback to the user.
* Generate Invoice: Creates a digital invoice for the completed purchase.

**7. Reviews & Ratings**

Enables user-generated content for products.

* Submit Product Review: Allows customers to write and submit reviews and assign ratings to purchased products.
* View Product Reviews: Displays all submitted reviews and ratings for a particular product.

**8. Admin Order Management**

These functions are for administrators to oversee orders.

* View All Orders: Allows administrators to see a list of all placed orders.
* Update Order Status: Enables administrators to change the status of an order (e.g., from "pending" to "shipped").
* View Order Details (Admin): Provides administrators with comprehensive details of any order.
* Process Refunds: Allows administrators to initiate refunds for returned or canceled orders.

**Non-Functional Considerations**

While not part of a functional analysis, it's important to acknowledge that the system must also meet non-functional requirements such as:

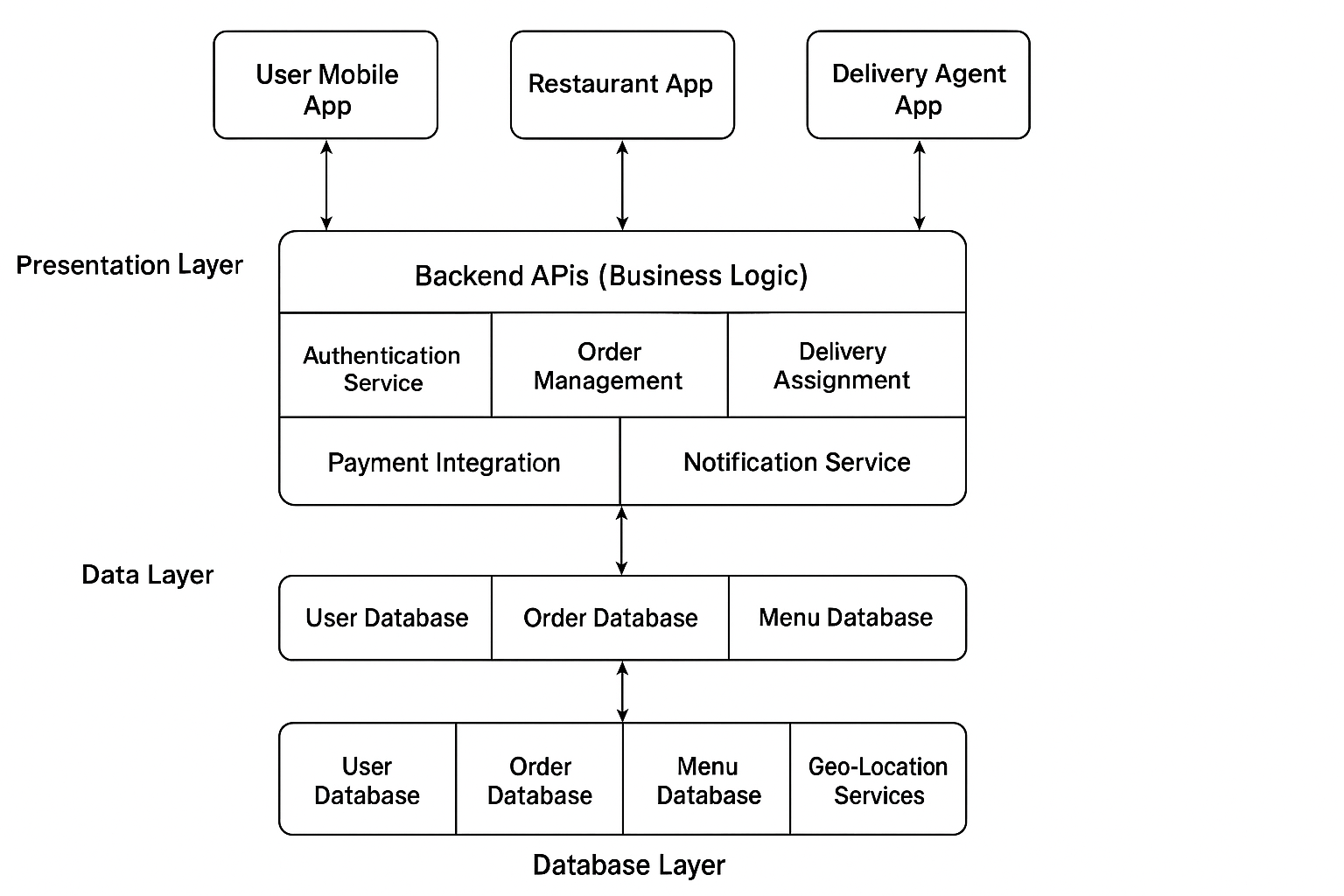
* Performance: The system should respond quickly to user requests.
* Security: User data and payment information must be protected.
* Usability: The system should be easy and intuitive for users to navigate.
* Scalability: The system should be able to handle an increasing number of users and products.
* Reliability: The system should be available and stable with minimal downtime.

This functional analysis provides a comprehensive overview of what an online shopping system is expected to do, forming the basis for subsequent design and development phases.

**-----------------------------------------------------------------------------------------------------------------**

**System Design**

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



**🍔 System Architecture: Food Delivery App**

**✅ 1. Overview**

The Food Delivery App allows users to **browse restaurants**, **place food orders**, **make payments**, and **track delivery** in real-time. The system also enables restaurants to manage menus and delivery agents to fulfill orders.

**🧱 2. System Architecture Components**

**🎨 A. Presentation Layer (Frontend / Mobile App)**

* **User App** (for Customers): Browse, order, pay, track delivery
* **Restaurant App**: Manage menu, accept orders
* **Delivery App**: Accept delivery tasks, update delivery status
* **Admin Web Panel**: Monitor activity, manage users/restaurants

**⚙️ B. Application Layer (Backend APIs / Server Logic)**

* **Authentication Service**: Login, signup, password reset
* **Order Management**: Cart, checkout, order processing
* **Menu & Restaurant Management**: Add/edit food items, pricing
* **Delivery Assignment**: Match orders to delivery agents
* **Payment Integration**: UPI/Wallet/Card payments
* **Notification Service**: Email/SMS/App alerts

**💾 C. Data Layer (Database & Storage)**

* **User Database**: Stores customer, restaurant, and delivery data
* **Order Database**: Tracks order history, status, payments
* **Menu Database**: Restaurant menus and item info
* **Geo-Location Services**: Store and track real-time delivery location

**🧩 4. Actors and Their Interfaces**

| **Actor** | **Interface** | **Functions** |
| --- | --- | --- |
| Customer | Mobile App | Browse, order, pay, track food |
| Restaurant | Restaurant App/Web | Manage menu, accept orders |
| Delivery Partner | Delivery App | Accept and deliver orders |
| Admin | Web Dashboard | Manage platform-wide operations |

**--------------------------------------------------------------------------------------------------------------------------**

**Software Testing**

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

**Test Case Specification for Simple Calculator**

| **Test Case ID** | **Test Scenario** | **Input(s)** | **Expected Output** | **Actual Output** | **Status (Pass/Fail)** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| TC\_001 | Add two positive integers | 5 + 3 | 8 | - | - | Basic addition |
| TC\_002 | Add a positive and a negative number | -4 + 6 | 2 | - | - | Mixed sign addition |
| TC\_003 | Add two negative numbers | -5 + (-3) | -8 | - | - | Negative addition |
| TC\_004 | Add two decimal numbers | 2.5 + 3.1 | 5.6 | - | - | Floating-point addition |
| TC\_005 | Subtract two integers | 10 - 4 | 6 | - | - | Basic subtraction |
| TC\_006 | Subtraction resulting in negative number | 4 - 7 | -3 | - | - | Negative result |
| TC\_007 | Multiply two positive integers | 3 \* 5 | 15 | - | - | Multiplication |
| TC\_008 | Multiply with zero | 9 \* 0 | 0 | - | - | Zero multiplication |
| TC\_009 | Multiply decimals | 1.5 \* 2.0 | 3.0 | - | - | Floating-point multiplication |
| TC\_010 | Divide two integers | 8 / 2 | 4.0 | - | - | Basic division |
| TC\_011 | Divide with result in decimal | 5 / 2 | 2.5 | - | - | Fractional result |
| TC\_012 | Division by zero | 6 / 0 | Error / Exception handled | - | - | Division by zero check |
| TC\_013 | Use of invalid operator | 5 & 3 | Error: Invalid operator | - | - | Operator validation |
| TC\_014 | Use of parentheses in expression | (4 + 5) \* 2 | 18 | - | - | Operator precedence |
| TC\_015 | Multiple operations in one expression | 2 + 3 \* 4 | 14 | - | - | Order of operations |
| TC\_016 | Expression with whitespaces | 7 + 2 | 9 | - | - | Ignore extra spaces |
| TC\_017 | Empty input |  | Error: Empty input | - | - | Input validation |
| TC\_018 | Input with only operator | + | Error: Incomplete expression | - | - | Input validation |
| TC\_019 | Very large numbers | 99999999 + 99999999 | 199999998 | - | - | Check for integer overflow handling |
| TC\_020 | Invalid characters | 3 + a | Error: Invalid character | - | - | Type checking |

**Assumptions**

* The calculator accepts input as either **single expressions (e.g., 5+3)** or **two values with operator (e.g., input1, operator, input2)**.
* It should handle **integer and float types**.
* The calculator should **gracefully handle invalid input**, not crash.
* If implemented via GUI, button input should be tested separately.

**--------------------------------------------------------------------------------------------------------------------------**

**Maintenance**

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

**🛠️ A Real-World Case of Critical Software Maintenance**

**✅ 1. Case Study Title:**

**Critical Maintenance of WhatsApp After a Global Outage (2022)**

**2. Background:**

**WhatsApp**, the popular messaging app owned by **Meta**, experienced a **major global outage on October 25, 2022**. Millions of users worldwide were unable to send or receive messages for over two hours. The outage affected both **individual and group messaging**, with services inaccessible across Android, iOS, and Web platforms.

**3. Nature of the Problem:**

* The core messaging system failed to connect with WhatsApp servers.
* Backend infrastructure likely experienced a **synchronization failure or deployment bug**.
* Authentication, message delivery, and real-time updates were completely non-functional.
* Internal services failed to route traffic efficiently due to configuration issues.

**4. Type of Maintenance Required:**

| **Type** | **Explanation** |
| --- | --- |
| **Corrective Maintenance** | Fixing bugs or errors that caused system-wide disruption |
| **Emergency Maintenance** | Immediate fix due to complete failure of core functionality (messaging) |
| **Configuration Maintenance** | Likely needed to restore settings and load balancers across regions |

**🛠️ 5. Maintenance Actions Taken:**

* Engineers immediately started **root-cause analysis** via logs and health-check monitors.
* Deployed a **patch to restore server communication protocols**.
* **Scaled backend infrastructure** to re-balance traffic loads globally.
* Verified services using real-time monitoring tools before restarting access.
* Released an **official apology and update** confirming issue resolution.

**6. Impact Before and After Maintenance:**

| **Aspect** | **Before Maintenance** | **After Maintenance** |
| --- | --- | --- |
| Messaging | Completely non-functional | Restored worldwide |
| User Experience | Users unable to communicate | Full service availability |
| Business Dependence | Customer service and businesses disrupted | Functionality restored within 2 hours |
| Trust in App | Temporary dip in confidence | Recovered after transparency from Meta |

**7. Lessons Learned:**

* Importance of **real-time monitoring** and alert systems
* Need for **redundant failover mechanisms** in cloud infrastructure
* Value of **clear communication** with users during critical downtime
* Regular **regression testing** post-deployment to avoid bugs in live systems

**--------------------------------------------------------------------------------------------------------------------------**

**30. DFD (Data Flow Diagram)**

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

**Data Flow Diagram- Hospital management system**

****

| **Entity/Process** | **Type** | **Description** |
| --- | --- | --- |
| **Patient** | External Entity | Registers in the system and receives diagnosis and reports. |
| **Doctor** | External Entity | Receives patient reports; provides diagnosis and prescriptions. |
| **Receptionist** | External Entity | Manages appointment requests, receives billing and reports info. |
| **Administrator** | External Entity | Monitors billing and prescriptions; manages internal operations. |
| **Patient Registration** | Process | Takes patient details and creates a unique record in the system. |
| **Appointment Scheduling** | Process | Handles booking of appointments requested by patients or managed by receptionist. |
| **Diagnosis & Treatment** | Process | Doctor provides medical advice, diagnoses conditions, and writes prescriptions. |
| **Billing & Payment** | Process | Calculates cost based on treatment and services; generates bills. |

**Data Flows**

| **From** | **To** | **Data Flow** |
| --- | --- | --- |
| Patient | Patient Registration | Registration Info |
| Receptionist | Appointment Scheduling | Appointment Requests |
| Diagnosis | Administrator | Prescriptions |
| Billing | Administrator | Billing Info |
| System | Doctor | Reports |
| System | Receptionist | Reports, Billing Info |

**--------------------------------------------------------------------------------------------------------------------------**

**31. Desktop Application**

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

**🧮 Desktop Application – Simple Calculator using GUI (Tkinter)**

**✅ Objective:**

Create a desktop-based calculator with basic arithmetic functions (**Add, Subtract, Multiply, Divide**) using a GUI framework.

**🛠️ Technology Stack:**

* **Programming Language:** Python
* **GUI Library:** Tkinter (built-in with Python)

**Features to Implement:**

* User-friendly number buttons (0–9)
* Arithmetic operators (+, –, ×, ÷)
* Equal (=) button to evaluate expressions
* Clear (C) button to reset input
* Error handling (e.g., divide by zero)

**Source Code (Python + Tkinter):**

import tkinter as tk

def click(event):

current = str(entry.get())

entry.delete(0, tk.END)

entry.insert(0, current + str(event.widget["text"]))

def clear():

entry.delete(0, tk.END)

def calculate():

try:

result = eval(entry.get())

entry.delete(0, tk.END)

entry.insert(0, str(result))

except Exception as e:

entry.delete(0, tk.END)

entry.insert(0, "Error")

# Create window

root = tk.Tk()

root.title("Simple Calculator")

root.geometry("300x400")

# Entry field

entry = tk.Entry(root, font=("Arial", 20), justify="right", bd=10)

entry.pack(fill=tk.BOTH, ipadx=8, ipady=8)

# Button layout

buttons = [

['7', '8', '9', '/'],

['4', '5', '6', '\*'],

['1', '2', '3', '-'],

['0', 'C', '=', '+']

]

for row in buttons:

frame = tk.Frame(root)

frame.pack(expand=True, fill="both")

for btn in row:

b = tk.Button(frame, text=btn, font=("Arial", 18), relief="ridge")

b.pack(side="left", expand=True, fill="both")

if btn == '=':

b.config(command=calculate)

elif btn == 'C':

b.config(command=clear)

else:

b.bind("<Button-1>", click)

root.mainloop()

**--------------------------------------------------------------------------------------------------------------------------**

**32. Flow Chart**

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

