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

**C languages**

**#include <stdio.h>**

**int main()**

**{**

**printf("Hello World\n");**

**return 0;**

**}**

**C++**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**cout << "Hello World" << endl;**

**return 0;**

**}**

**Compare :-**

| **Feature** | **C** | **C++** |
| --- | --- | --- |
| **Code length** | **Short and simple** | **Slightly longer due to cout syntax** |
| **Input/Output** | **printf** | **cout** |
| **Library** | **<stdio.h>** | **<iostream>** |
| **Namespace** | **Not required** | **using namespace std; needed for cout** |
| **OOP support** | **No native support (procedural)** | **Supports classes and OOP** |
| **Syntax** | **Procedural style** | **Can be procedural or object-oriented** |

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

**When a client communicates with a server data travels through several layers and components.**

1. **Client Request**

* **The client creates a request (like opening a webpage or sending a form).**
* **This request is processed by the application layer (HTTP/HTTPS).**

1. **Transport Layer**

* **The request is broken into packets by the TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).**
* **TCP ensures reliable delivery; UDP is faster but less reliable.**

1. **Network Layer**

* **Packets are addressed using IP (Internet Protocol).**
* **The IP layer determines the path to the server using routers.**

1. **Physical/Data Link Layer**

* **Data is converted to electrical signals, radio waves, or optical signals to travel over physical media (like fiber optics, Wi-Fi, or Ethernet cables).**

1. **Internet Infrastructure**

* **Packets pass through multiple routers and switches across the Internet.**
* **Each router reads the destination IP and forwards the packet toward the server.**

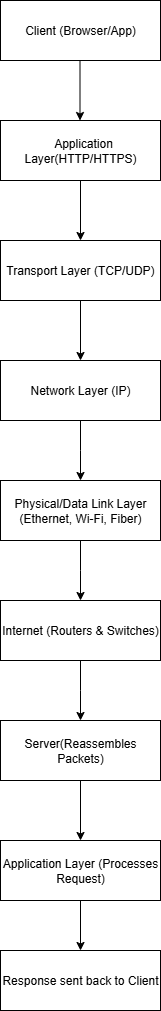
1. **Server Reception**

* **The server receives packets, and the TCP layer reassembles them.**
* **The server’s application layer processes the request (like generating a webpage response).**

1. **Response Back to Client**

* **The server sends the response using the same layers in reverse.**
* **The client receives the data and displays it (e.g., a web page in the browser).**

**Diagram:**

****

**Lab – 3: Design a simple HTTP client-server communication in any language**

1. **Client Request**

* **The client creates a request (like opening a webpage or sending a form).**
* **This request is processed by the application layer (HTTP/HTTPS).**

1. **Transport Layer**

* **The request is broken into packets by the TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).**
* **TCP ensures reliable delivery; UDP is faster but less reliable.**

1. **Network Layer**

* **Packets are addressed using IP (Internet Protocol).**
* **The IP layer determines the path to the server using routers.**

1. **Physical/Data Link Layer**

* **Data is converted to electrical signals, radio waves, or optical signals to travel over physical media (like fiber optics, Wi-Fi, or Ethernet cables).**

1. **Internet Infrastructure**

* **Packets pass through multiple routers and switches across the Internet.**
* **Each router reads the destination IP and forwards the packet toward the server.**

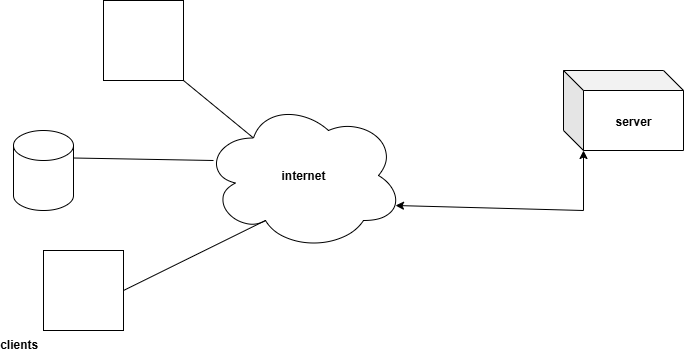
1. **Server Reception**

* **The server receives packets, and the TCP layer reassembles them.**
* **The server’s application layer processes the request (like generating a webpage response).**

1. **Response Back to Client**

* **The server sends the response using the same layers in reverse.**
* **The client receives the data and displays it (e.g., a web page in the browser).**

**Diagram :**

****

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

**1.Broadband (DSL / Cable Internet)**

**Pros:**

* **Widely available in cities and towns.**
* **Relatively affordable compared to newer technologies.**
* **Stable connection for browsing, streaming, and basic work.**
* **Always-on connection (no need to dial in).**

**Cons:**

* **Speed depends on distance from the provider’s infrastructure.**
* **It can slow down during peak hours.**
* **Not as fast as fiber-optics.**

**2. Fiber-Optic Internet**

**Pros:**

* **Extremely fast speeds (can reach 1 Gbps or higher).**
* **Low latency, great for gaming and video conferencing.**
* **More reliable with less interference.**
* **Can handle heavy usage without slowdown.**

**Cons:**

* **Limited availability, especially in rural areas.**
* **More expensive to install and maintain.**
* **May require new wiring in older buildings.**

**3. Satellite Internet**

**Pros:**

* **Available in remote and rural areas where other types are not.**
* **Does not require cables or phone lines.**
* **Good for basic browsing and communication.**

**Cons:**

* **High latency (signal travels to space and back).**
* **Weather can affect connection quality.**
* **Lower data limits and higher cost compared to broadband.**

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

**1. Simulating HTTP Requests**

**GET request (most common)**

**curl https://example.com**

**GET request with headers**

**curl -i** [**https://example.com**](https://example.com)

**Download a file**

**curl -O** [**https://example.com/file.zip**](https://example.com/file.zip)

**2. Simulating FTP Requests**

**Anonymous FTP download**

**curl** [**ftp://ftp.example.com/file.txt -o file.txt**](ftp://ftp.example.com/file.txt%20-o%20file.txt)

**FTP download with username/password**

**curl -u myuser:mypassword** [**ftp://ftp.example.com/file.txt -o file.txt**](ftp://ftp.example.com/file.txt%20-o%20file.txt)

**Upload a file via FTP**

**curl -T upload.txt -u myuser:mypassword** [**ftp://ftp.example.com/**](ftp://ftp.example.com/)

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

1. **SQL Injection (SQLi)**

**Explanation:  
Occurs when user input is directly included in SQL queries without proper sanitization. An attacker can insert malicious SQL commands to read, modify, or delete database data.  
Solution:**

* **Use prepared statements / parameterized queries instead of string concatenation.**
* **Validate and sanitize all user inputs.**
* **Limit database permissions to the least privilege necessary.**

1. **Cross-Site Scripting (XSS)**

**Explanation:  
Occurs when malicious scripts are injected into a web application and executed in other users’ browsers. This can steal cookies, session tokens, or manipulate the page content.**

**Solution:**

* **Escape all user-generated output before rendering in HTML.**
* **Use frameworks or template engines that auto-escape content.**
* **Implement Content Security Policy (CSP) to block unauthorized scripts.**

1. **Cross-Site Request Forgery (CSRF)**

**Explanation:  
Tricks an authenticated user into unintentionally performing actions (like changing their password or transferring funds) on a web application without their knowledge.**

**Solution:**

* **Use anti-CSRF tokens in forms and requests.**
* **Require re-authentication for sensitive operations.**
* **Set cookies with the Same Site attribute to prevent cross-site sending.**

**Lab-7 *:* Identify and classify 5 applications you use daily as either system software or application software**

1. **Windows 10/11 – This is system software because it is an operating system that controls hardware and allows applications to run.**
2. **Google Chrome – This is application software because it is used to browse and access websites.**
3. **Microsoft Word – This is application software because it is used to create and edit documents.**
4. **Antivirus program (e.g., Avast) – This is system software because it protects the system by detecting and removing malware.**
5. **Spotify – This is application software because it is used to stream music and podcasts.**

**Lab-8:- : Design a basic three-tier software architecture diagram for a web application**

A white rectangular object with black text

AI-generated content may be incorrect.

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

**A Hotel Booking Management System allows customers to search for rooms, check availability, and make bookings online. The system follows a three-tier architecture to ensure scalability, maintainability, and separation of concerns:**

**Presentation Layer – User interface for interacting with the system.**

**Business Logic Layer (BLL) – Handles rules, processes, and core operations.**

**Data Access Layer (DAL) – Manages communication with the database.**

**1. Presentation Layer (UI Layer)**

**Functionality:**

* **Provides an interface for customers and hotel staff to interact with the system.**
* **Collects inputs (e.g., check-in/check-out dates, room type).**
* **Displays outputs (e.g., available rooms, booking confirmation).**

**Example in System:**

* **Customer Side: A user logs in, selects travel dates, chooses a room, and confirms payment.**
* **Staff Side: Hotel staff log in to update room availability or modify reservations.**

**Technologies Used:**

* **Web: HTML, CSS, JavaScript, React.js / Angular.**
* **Mobile: Flutter / React Native.**

**2. Business Logic Layer**

**Functionality:**

* **Validates user input (e.g., check-out date must be after check-in date).**
* **Calculates total booking cost (including taxes and discounts).**
* **Enforces business rules (e.g., maximum stay limit, cancellation policies).**
* **Communicates with the Data Access Layer to retrieve/update information.**

**Example in System:**

* **Check if requested room is available for given dates.**
* **Applies for promotional discount if the user is a loyalty member.**
* **Cancel booking and initiates refund according to policy.**

**Technologies Used:**

* **Backend Frameworks: Java (Spring Boot), Node.js, or .NET Core.**
* **Written in languages like Java, C#, or JavaScript.**

**3. Data Access Layer (DAL)**

**Functionality:**

* **Manages all database operations (CRUD – Create, Read, Update, Delete).**
* **Uses SQL queries or ORM (Object-Relational Mapping) to interact with the database.**
* **Ensures secure data handling (prevents SQL injection, applies encryption).**

**Example in System:**

* **Retrieves available room data from the Rooms table.**
* **Insert new booking details into the Bookings table.**
* **Updates room status to "booked" after a successful reservation.**

**Technologies Used:**

* **Databases: MySQL, PostgreSQL, MongoDB.**
* **ORM Tools: Hibernate, Entity Framework, Sequalae.**

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

**Part 1: Exploring Software Environments**

**1. Development Environment**

* **Purpose: Used by developers to write and test code locally.**
* **Features:**
* **Code editors/IDEs (e.g., VS Code, IntelliJ)**
* **Debugging tools**
* **Local database or mock APIs**
* **Example: Running a Flask server on your laptop to test changes before committing code.**

**2. Testing Environment (Test / QA)**

* **Purpose: Used to verify that the application works as intended, often by a QA team.**
* **Features:**
* **Same configurations as production, but with dummy/test data**
* **Automated test scripts (e.g., Selenium, JUnit)**
* **Bug tracking integration**
* **Example: QA engineers run UI and API tests here before approving release.**

**3. Production Environment (Prod)**

* **Purpose: Live system accessible to real users.**
* **Features:**
* **High performance and security settings**
* **Real data**
* **Monitoring tools (e.g., Grafana, Prometheus)**
* **Example: The live website your customers use.**

**Part 2: Setting Up a Basic Environment in a Virtual Machine**

* **Open VirtualBox / VMware.**
* **Click New → Name your VM (e.g., "Ubuntu Dev Env").**
* **Select the OS type (Linux → Ubuntu 64-bit).**

**Allocate:**

* **RAM: At least 2 GB (4 GB recommended)**
* **CPU: 2 courses if possible**

**Create a virtual hard disk:**

* **Type: VDI (VirtualBox Disk Image)**
* **Size: 20 GB or more**
* **Storage: Dynamically allocated**

**Lab-11:- Write and upload your first source code file to GitHub*.***

1. **Create your first code file**

* **Open Notepad, VS Code, or any text editor.**
* **Type the following sample Python code:**

**python**

**# hello.py**

**print("Hello, GitHub! This is my first uploaded code.")**

* **Save the file as hello.py on your computer (e.g., in Documents or Desktop).**

1. **Create a GitHub repository**

* **Go to** [**https://github.com**](https://github.com) **and log in.**
* **Click the + icon in the top right → New repository.**
* **Enter a repository name, e.g., my-first-code.**
* **Keep it Public (so anyone can see it).**
* **Click Create repository.**

1. **Upload your file to GitHub**

**Directly on the website (no Git commands)**

* **Inside your new repository, click Add file → Upload files.**
* **Drag and drop your hello.py file.**
* **Scroll down, add a commit message (e.g., *"Add my first code file"*).**
* **Click Commit changes.**

**Lab- 12 Create a GitHub repository and document how to commit and push code changes**

1. **Install GitHub Desktop**

* **Download from: https://desktop.github.com**
* **Install and sign in with your GitHub account.**

1. **Create a New Repository**

* **Open GitHub Desktop.**
* **Go to File → New Repository.**
* ***Fill in:***
* **Name: e.g., my-first-repo**
* **Description: Optional**
* **Local path: Where the repo will be stored on your computer**
* **Initialize with:**

***✔* README**

**✔ Git ignore (choose a template if needed)**

* **Click Create Repository.**

1. **Publish to GitHub**

* **After creating the repo locally, click Publish repository (top right).**
* **Choose:**
* **Name (same or different)**
* **Public or Private**
* **Click Publish Repository.**

1. **Make Changes & Commit**

* **Open the repository’s folder in your editor (e.g., VS Code) and make changes (create/edit files).**
* **Go back to GitHub Desktop — you’ll see Changed files listed.**
* **At the bottom left:**
* **Summary: Write a short commit message (e.g., "Add new feature").**
* **Description: Optional longer explanation.**
* **Click Commit to main.**

1. **Push Changes to GitHub**

* **After committing, click Push origin (top bar) to upload your changes to GitHub.**

1. **Pulling Changes (if working with others)**

* **Click Fetch origin to get updates from GitHub.**
* **If updates exist, click Pull origin*.***

***.***

**Lab – 13 Create a student account on GitHub and collaborate on a small project with a classmate**

1. **Create a GitHub Account**

* **Go to** [**github.com**](https://github.com)**.**
* **Click Sign up.**
* **Enter:**
* **Username (unique)**
* **Email address**
* **Password**
* **Verify your account (puzzle or email confirmation).**
* **Choose Free plan.**

1. **Apply for GitHub Student Developer Pack (Optional but Recommended)**

**The Student Pack gives free tools, hosting, and premium features.**

* **Go to education.github.com/pack.**
* **Click Get your pack.**
* **Verify you are a student:**
* **Provide school-issued email or upload student ID.**
* **Wait for approval (usually within 1–3 days).**

1. **Create a Small Project Repository**

* **Log in to GitHub.**
* ***Click New repository.***
* ***Fill details:***
* **Repository name (e.g., student-collab-project)**
* **Description (short summary)**
* **Public or Private (Public for open collaboration)**
* **Check Add a README file.**
* **Click Create repository*.***

1. **Invite Your Classmate to Collaborate**

* **Go to your repository.**
* **Click Settings → Collaborators.**
* **Click Add people.**
* **Enter your classmate’s GitHub username or email.**
* **Click Add collaborator.**
* **Your classmate will accept the invitation from their email.**

1. **Collaboration on the Project**

**Option A – Using GitHub Website**

* **Both can directly edit files in the browser.**
* **Click Add file → Create new file or Upload files.**

**Option B – Using GitHub Desktop**

* **Download GitHub Desktop.**
* **Sign in with your GitHub account.**
* **Clone the repository to your computer.**
* **Make changes locally and Commit → Push origin to update GitHub.**
* **Your classmate can Pull to get the latest changes.**

1. **Workflow for Collaboration**

* **Pull latest changes before editing.**
* **Make edits or add files.**
* **Commit changes with a short message.**
* **Push changes to GitHub.**
* **Classmate repeats the same to sync work.**

**Lab-14: Create a list of software you use regularly and classify them into the following**

**categories: system, application, and utility software**

1. **System Software (runs and manages computer hardware, provides platform for other software)**

* **Microsoft Windows 10/11 (Operating System)**
* **macOS (Operating System)**
* **Android OS / iOS**
* **Linux Ubuntu**

1. **Application Software (programs designed for specific user tasks)**

* **Microsoft Word (word processing)**
* **Google Chrome (web browser)**
* **WhatsApp (messaging)**
* **Spotify (music streaming)**
* **Zoom (video conferencing)**

1. **Utility Software (helps maintain, optimize, or secure the system)**

* **WinRAR / 7-Zip (file compression)**
* **CCleaner (system cleanup)**
* **Windows Defender (antivirus)**
* **Disk Cleanup Tool (storage management)**
* **Backup & Restore Tool**

**Lab 15:- Follow a GIT tutorial to practice cloning, branching, and merging repositories.**

1. **Clone a Repository:-First, we copy an existing repository to your computer.  
   You can either clone your own repo or a public one.**

**# Go to the folder where you want to store the project**

**cd path/to/your/folder**

**# Clone from GitHub (replace with your repo link)**

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

**# Enter the project folder**

**cd repo-name**

1. **Create a New Branch:-Branches let you work on changes without affecting the main code.**

**# Check existing branches**

**git branch**

**# Create a new branch**

**git branch feature-branch**

**# Switch to the new branch**

**git checkout feature-branch**

**# OR shortcut to create & switch:**

**git checkout -b feature-branch**

1. **Make Changes and Commit:-Edit a file in the project (e.g., index.html or README.md)  
   Then save and commit it.**

**# Stage changes**

**git add .**

**# Commit changes**

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

1. **Merge Branch into Main:-Once you’re happy with the changes, merge them into the main branch.**

**# Switch back to main branch**

**git checkout main**

**# Merge feature branch into main**

**git merge feature-branch**

1. **Push Changes to GitHub**

**# Push main branch to GitHub**

**git push origin main**

1. ***Delete Branch (Optional)***

**# Delete locally**

**git branch -d feature-branch**

**# Delete from GitHub**

**git push origin --delete feature-branch**

**Lab-16 :- Write a report on the various types of application software and how they improve**

1. **productivity.**

* **Introduction:- Application software refers to computer programs designed to help users perform specific tasks, ranging from personal productivity to professional operations. Unlike system software, which manages hardware, application software enables end-users to accomplish work faster, more accurately, and with greater convenience.**

1. **Types of Application Software**
2. **Word Processing Software**

**Examples: Microsoft Word, Google Docs  
Purpose: Create, edit, and format text documents.  
Productivity Impact: Speeds up report writing, automates formatting, includes spelling/grammar checks, and allows easy sharing for collaboration.**

1. **Spreadsheet Software**

**Examples: Microsoft Excel, Google Sheets  
Purpose: Organize, calculate, and analyze numerical data.  
Productivity Impact: Automates calculations, creates visual charts, and manages large datasets efficiently.**

1. **Presentation Software**

**Examples: Microsoft PowerPoint, Canva  
Purpose: Create visually appealing slideshows for meetings, training, and pitches.  
Productivity Impact: Simplifies visual communication, enhances audience engagement, and saves preparation time with templates.**

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

**Examples: Microsoft Access, MySQL  
Purpose: Store, retrieve, and manage structured data.  
Productivity Impact: Reduces manual record-keeping, ensures data accuracy, and enables quick data queries.**

1. **Project Management Software**

**Examples: Trello, Asana, Microsoft Project  
Purpose: Plan, schedule, and monitor projects.  
Productivity Impact: Improves task coordination, tracks deadlines, and enhances team collaboration.**

1. **Communication & Collaboration Tools**

**Examples: Microsoft Teams, Slack, zoom  
Purpose: Facilitate real-time messaging, video conferencing, and file sharing.Productivity Impact: Eliminates delays in communication, supports remote work, and speeds up decision-making.**

1. **Graphic Design & Multimedia Software**

**Examples: Adobe Photoshop, CorelDRAW  
Purpose: Create and edit images, videos, and multimedia content.  
Productivity Impact: Accelerates content creation, improves visual appeal, and supports marketing efforts.**

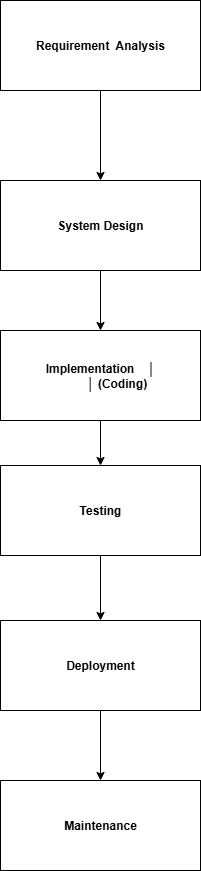
1. **Specialized Industry Software**

**Examples: AutoCAD (engineering), Tally ERP (accounting), MATLAB (research)  
Purpose: Perform industry-specific tasks.  
Productivity Impact: Enhances accuracy, saves time on repetitive processes, and meets specialized needs efficiently*.***

1. **How Application Software Improves Productivity**

* **Automation of Tasks: Reduces manual work through features like formulas, macros, and AI suggestions.**
* **Time Efficiency: Quick processing of data and documents.**
* **Collaboration: Enables multiple users to work on projects simultaneously.**
* **Accuracy: Minimizes human errors through built-in error-checking tools.**
* **Accessibility: Cloud-based apps allow work from anywhere, anytime.**

**Lab 17: Create a flowchart representing the Software Development Life Cycle (SDLC)**



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

1. **Introduction**

* **Purpose:- The purpose of the Library Management System (LMS) is to automate and manage the daily operations of a library, including book cataloging, member registration, book lending, and returns. It aims to replace manual record-keeping with a centralized, efficient, and user-friendly system.**
* **Scope:- The LMS will allow librarians to manage books and members, track issued and returned books and maintain fine records for overdue books. The system will be accessible only to authorized staff.**

1. **Functional Requirements**

**Book Management**

* **Add new books with details (title, author, ISBN, category, quantity).**
* **Update book details.**
* **Remove books from the system.**
* **Search for books by title, author, or category.**

**Member Management**

* **Register new members with details (name, address, contact number, ID).**
* **Update member details.**
* **Remove member records.**
* **Search for members by name or ID.**

**Lending & Returns**

* **Issue books to members (with issue date and due date).**
* **Return books and update stock.**
* **Calculate and record fines for overdue returns.**

**Reporting**

* **Generate daily, weekly, or monthly reports for issued and returned books.**
* **View fine collection reports.**

1. **Non-Functional Requirements**

* **Usability: The system should have an easy-to-use interface for staff.**
* **Performance: Search operations should return results within 2 seconds.**
* **Security: Only authorized users can log in.**
* **Data Backup: System should allow regular backup of records.**

1. **User Roles**

* **Administrator: Full access to all features.**
* **Librarian: Can manage books, members, and issue/return operations.**

1. **Constraints**

* **The system will be a desktop application running on Windows OS.**
* **Database should be SQLite or MySQL for storage.**

1. **Assumptions**

* **The library operates 6 days a week.**
* **Each member can borrow a maximum of 5 books at a time.**

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

1. **Objective:-The goal of the online shopping system is to allow customers to browse, search, select, and purchase products through the internet, while enabling administrators and sellers to manage product listings, orders, and customers efficiently.**
2. **Main Functional Areas**

**User Management**

* **Customer Registration & Login:**
* **Create new accounts with personal details (name, email, phone, address).**
* **Login using email/username and password.**
* **Password recovery/reset functionality.**
* **Profile Management:**
* **Edit personal details.**
* **View past orders and transaction history.**

**Product Management**

* **Catalog Browsing:**
* **View products by categories, subcategories, and brands.**
* **Apply filters (price range, rating, availability).**
* **Product Search:**
* **Keyword-based search.**
* **Advanced search by attributes (size, color, etc.).**
* **Product Details:**
* **View product descriptions, images, price, specifications, and reviews.**

**Shopping Cart & Wishlist**

* **Add/remove products to/from cart.**
* **Update product quantities in the cart.**
* **Save products in a Wishlist for later purchase.**

**Order Management**

* **Placing Orders:**
* **Select products from cart, choose shipping method, and make payment.**
* **Order Tracking:**
* **View current status (processing, shipped, delivered).**
* **Order History:**
* **View past orders and details.**

**Payment Processing**

* **Support multiple payment methods (credit/debit card, UPI, net banking, cash on delivery).**
* **Secure payment gateway integration.**
* **Generate invoices after successful payment.**

**Review & Rating System**

* **Customers can leave ratings and feedback for purchased products.**
* **Display average ratings for each product.**

**Admin/Seller Functions**

* **Add/edit/remove product listings.**
* **Manage categories and subcategories.**
* **View and process customer orders.**
* **Handle customer queries and complaints.**
* **Generate sales reports and analytics.**

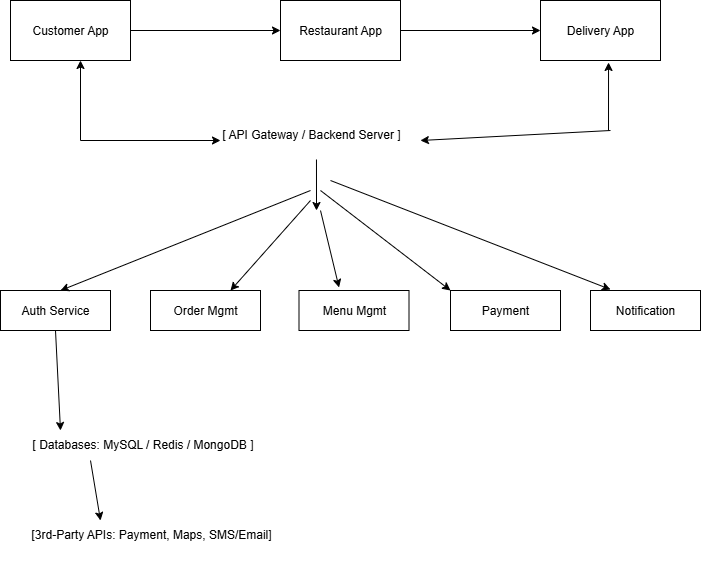
**3. System Interaction Diagram (High-Level)**

**Actors:**

* **Customer → Browse → Select → Pay → Track Order.**
* **Admin/Seller → Manage Products → Process Orders → View Reports.**

**Payment Gateway → Verify and process payments.**

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



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

| **Test Case ID** | **Test Description** | **Input** | **Expected Output** | **Remarks** |
| --- | --- | --- | --- | --- |
| **TC01** | **Add two positive numbers** | **5 + 3** | **8** | **Basic addition** |
| **TC02** | **Add positive and negative number** | **10 + (-4)** | **6** | **Sign handling** |
| **TC03** | **Subtract two numbers** | **9 - 2** | **7** | **Basic subtraction** |
| **TC04** | **Subtract resulting negative number** | **4 - 9** | **-5** | **Negative result** |
| **TC05** | **Multiply two positive numbers** | **6 × 7** | **42** | **Basic multiplication** |
| **TC06** | **Multiply by zero** | **9 × 0** | **0** | **Zero property** |
| **TC07** | **Multiply by negative number** | **(-5) × 3** | **-15** | **Sign handling** |
| **TC08** | **Divide two positive numbers** | **20 ÷ 4** | **5** | **Basic division** |
| **TC09** | **Divide resulting in decimal** | **7 ÷ 2** | **3.5** | **Decimal handling** |
| **TC10** | **Divide by zero** | **5 ÷ 0** | **Error / Infinity** | **Error handling** |
| **TC11** | **Add decimal numbers** | **2.5 + 3.1** | **5.6** | **Floating point** |
| **TC12** | **Large number multiplication** | **100000 × 100000** | **10000000000** | **Large value handling** |
| **TC13** | **Sequential operations (no brackets)** | **5 + 3 × 2** | **11** | **Operator precedence** |
| **TC14** | **Sequential operations (with brackets)** | **(5 + 3) × 2** | **16** | **Bracket handling** |
|  |  |  |  |  |

**Lab 22: Document a real-world case where a software application required critical**

**maintenance.**

**Case Study: Boeing 787 Dreamliner – Lithium-Ion Battery Software Patch (2013)**

**1. Background**

**In January 2013, several Boeing 787 Dreamliner aircraft experienced overheating in lithium-ion batteries due to a software control issue in the battery monitoring system. The malfunction risked in-flight fires and prompted the FAA (Federal Aviation Administration) to ground the entire 787 fleet worldwide.**

**2. Problem Description**

* **The battery management software failed to correctly monitor and control charging cycles, leading to overcharging and overheating.**
* **There was no sufficient fail-safe mechanism in the control algorithm to detect early thermal runaway.**
* **This was classified as a critical maintenance situation because it directly affected passenger safety and regulatory compliance.**

**3. Type of Maintenance Performed**

**This case involved a Corrective & Preventive Maintenance approach:**

1. **Corrective Maintenance – Immediate software patch to fix the faulty charging control logic.**
2. **Preventive Maintenance – Added extra safety thresholds, improved temperature monitoring, and implemented early warning alerts to prevent recurrence.**

**4. Actions Taken**

* **Updated the battery management software to:**
  + **Monitor temperature at multiple points.**
  + **Shut down charging automatically when safe limits were exceeded.**
  + **Log battery performance data for predictive analysis.**
* **Conducted rigorous ground testing before redeploying the aircraft.**
* **Worked with FAA to validate changes and lift the grounding order.**

**5. Outcome**

* **The software update eliminated the immediate overheating issue.**
* **Fleet returned to service after three months of grounding.**
* **Boeing also redesigned the battery containment system for added safety.**

**6. Lessons Learned**

* **Real-time safety-critical systems must have multiple fail-safes.**
* **Regular predictive maintenance updates can prevent emergency grounding.**
* **Software in embedded systems (like aircraft) must undergo extreme scenario testing before release**

**Lab -23 Create a DFD for a hospital management system**

**O level :**



**1 level :**

A diagram of a medical system

AI-generated content may be incorrect.

**Level 2**

A black background with white ovals and black text

AI-generated content may be incorrect.

**Lab 24 : Build a simple desktop calculator application using a GUI library**

**1. Plan the Features**

* **Basic arithmetic: addition, subtraction, multiplication, division.**
* **Clear function to reset input.**
* **Display area to show entered numbers and results.**
* **Buttons for digits (0–9), decimal point, and operators (+, −, ×, ÷, =).**

**2. Choose a GUI Library**

* **Python: Tkinter, PyQt, Kivy.**
* **Java: Swing, JavaFX.**
* **C#: Windows Forms, WPF.  
  Pick the one you’re most comfortable with.**

**3. Design the Layout**

* **Top area: Text/display box to show the current input or result.**
* **Middle area: Buttons for numbers (arranged in a grid like a calculator).**
* **Side area or bottom row: Operator buttons (+, −, ×, ÷, =) and Clear (C) button.**

**Example layout:**

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

**| Display Area |**

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

**| 7 | 8 | 9 | ÷ |**

**| 4 | 5 | 6 | × |**

**| 1 | 2 | 3 | − |**

**| 0 | . | = | + |**

**| Clear Button |**

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

**4. Implement the Logic**

* **When a button is clicked, append its value to the display.**
* **When “=” is clicked, evaluate the mathematical expression and show the result.**
* **When “C” is clicked, clear the display.**

**5. Style & Test**

* **Adjust font sizes and button spacing for a clean look.**
* **Test all combinations to ensure calculations are correct.**
* **Handle errors (like dividing by zero) by showing a friendly message.**

**Lab 25 Draw a flowchart representing the logic of a basic online registration system**

