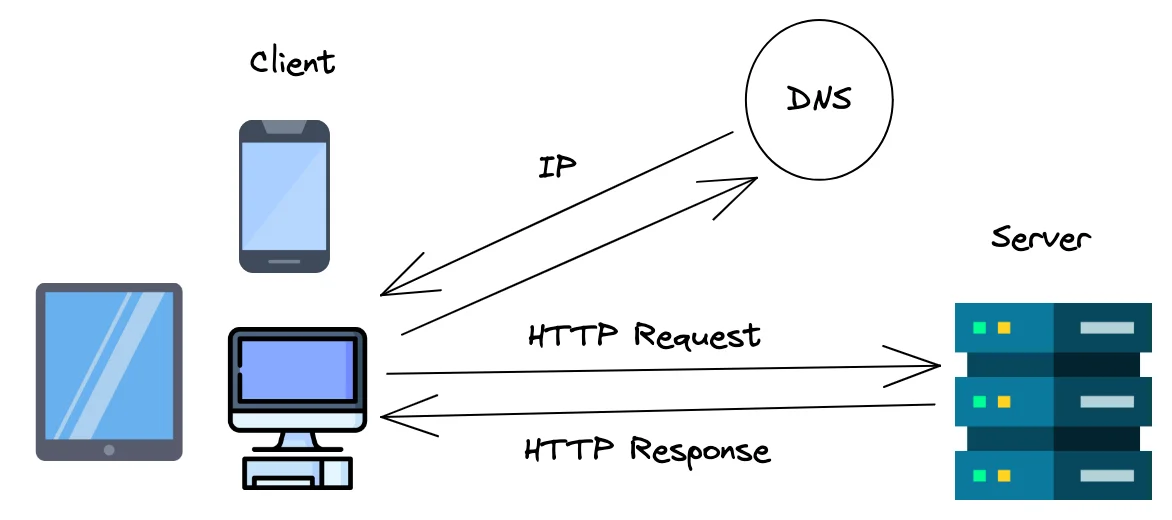
LAB EXERCISE

# World Wide Web & How Internet Works

**Q2. Research and create a diagram of how data is transmitted from a client to a server over the internet.**

**A.**



1. **Client initiates a request:** User types a URL in a browser.
2. **Router forwards the request:** Local network sends it to the ISP.
3. **ISP routes the packet:** Data packets are forwarded through various routers over the internet.
4. **DNS resolution occurs:** Domain name is translated into a server IP address.
5. **Server receives request:** Server processes the request and sends back the appropriate response (e.g., HTML page, data).
6. **Client receives response:** Browser renders the content.

# Network Layers on Client and Server

**Q3.** **Design a simple HTTP client-server communication in any language.**

**A.** Here's a simple **HTTP client-server communication** example using **Python** with the built-in http.server and http.client libraries. This example demonstrates how to:

1. Run a basic HTTP server that listens for GET requests.
2. Create a client that sends a GET request to the server.

**✅ Server Code (server.py)**

from http.server import BaseHTTPRequestHandler, HTTPServer

class SimpleHandler(BaseHTTPRequestHandler):

def do\_GET(self):

# Send response status code

self.send\_response(200)

# Send headers

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

self.end\_headers()

# Send the response body

message = "Hello from the server!"

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

def run(server\_class=HTTPServer, handler\_class=SimpleHandler, port=8000):

server\_address = ('', port)

httpd = server\_class(server\_address, handler\_class)

print(f"Server running on port {port}...")

httpd.serve\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

run()

**✅ Client Code (client.py)**

import http.client

def make\_request():

conn = http.client.HTTPConnection("localhost", 8000)

conn.request("GET", "/")

response = conn.getresponse()

print("Status:", response.status)

print("Response:", response.read().decode())

conn.close()

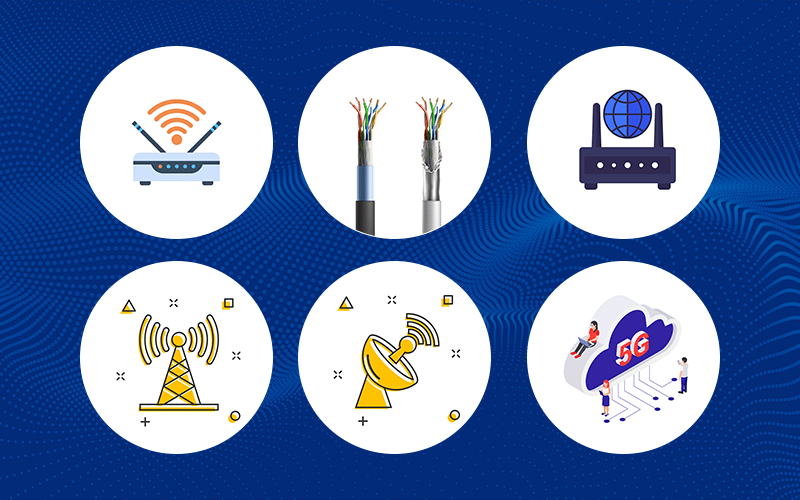
if \_\_name\_\_ == "\_\_main\_\_":

make\_request()

# **Types of Internet Connections**

**Q3. Research different types of internet connections (e.g., broadband, Fiber, satellite) and list their pros and cons**

**A.**



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

* **Description**: High-speed internet over telephone (DSL) or cable TV (cable) lines.

| **Pros** | **Cons** |
| --- | --- |
| Widely available | Speed can vary by location |
| Relatively affordable | DSL is slower than other options |
| Always on (no dial-up needed) | Shared bandwidth can cause slowdowns |

**2. Fiber Optic**

* **Description**: Internet through Fiber-optic cables that transmit data as light.

| **Pros** | **Cons** |
| --- | --- |
| Extremely fast speeds (up/down) | Limited availability in rural areas |
| Low latency | Higher installation cost |
| Ideal for streaming/gaming | Not always bundled with TV/phone plans |

**3. Satellite**

* **Description**: Internet via satellites orbiting Earth (good for rural/remote areas).

| **Pros** | **Cons** |
| --- | --- |
| Available almost everywhere | High latency (bad for gaming, VoIP) |
| Good for rural areas | Slower speeds than Fiber/cable |
| Doesn't require ground infrastructure | Weather can disrupt signal |

**4. Mobile (4G/5G)**

* **Description**: Wireless internet via cellular networks.

| **Pros** | **Cons** |
| --- | --- |
| Portable and flexible | Data caps and throttling |
| 5G can be very fast | Signal strength varies by location |
| No fixed installation needed | Can be expensive for high usage |

**5. Fixed Wireless**

* **Description**: Uses radio signals from a nearby tower to deliver internet.

| **Pros** | **Cons** |
| --- | --- |
| Good alternative in rural areas | Requires clear line-of-sight to tower |
| Faster than satellite in many cases | Speeds can be inconsistent |

# **Protocols**

**Q4.** **Simulate HTTP and FTP requests using command line tools (e.g., curl).**

**A.** Here's how you can **simulate HTTP and FTP requests** using the curl command from the command line.

**🌐 Simulate HTTP Requests with curl**

**🔸 1. Basic HTTP GET Request**

bash

curl http://example.com

This retrieves the contents of the webpage.

**🔸 2. HTTP GET with Headers**

bash

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

Adds a custom header (User-Agent) to the request.

**🔸 3. HTTP POST Request**

bash

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

Sends form data using POST.

**🔸 4. Download a File**

bash

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

Downloads a file and saves it with its original name.

**📁 Simulate FTP Requests with curl**

You can also use curl to interact with FTP servers.

✅ **Note**: Replace ftp.example.com with a real FTP server and provide correct credentials.

**🔸 1. FTP File Download**

bash

curl -u username:password ftp://ftp.example.com/path/to/file.txt -O

**🔸 2. List Directory on FTP Server**

bash

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

**🔸 3. Upload File to FTP Server**

bash

curl -T localfile.txt -u username:password <ftp://ftp.example.com/upload/>

# **Application Security**

**Q5. Identify and explain three common application security vulnerabilities. Suggest possible solutions.**

**A.**

**1. SQL Injection**

**Description**:  
Occurs when an attacker injects malicious SQL code into input fields to manipulate the backend database (e.g., extract or delete data).

**Solution**:

* Use **prepared statements** or **parameterized queries**
* Sanitize and validate all user inputs
* Limit database permissions for the application

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

**Description**:  
Allows attackers to inject malicious scripts (usually JavaScript) into web pages viewed by other users, potentially stealing session cookies or credentials.

**Solution**:

* **Escape and encode output** to prevent scripts from running
* Use **Content Security Policy (CSP)**
* Sanitize inputs and avoid inserting raw HTML from users

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

**Description**:  
Tricks a logged-in user’s browser into sending a request they didn’t intend, like changing account settings or making unauthorized transactions.

**Solution**:

* Use **anti-CSRF tokens** in forms
* Validate request origins (check Referrer or Origin headers)
* Require **authentication for sensitive actions**

# **Software Applications and Its Types**

**Q6.** **Identify and classify 5 applications you use daily as either system software or application software.**

**A.**

**✅ Application Software**

These are programs designed for end users to perform specific tasks.

1. **Google Chrome**   
   → **Application software**
2. **Microsoft Word**   
   → **Application software**
3. **Steam**   
   → **Application software**
4. **WhatsApp (Desktop/Mobile)**   
   → **Application software**
5. **Calculator**   
   → **Application software**

**🖥️ System Software**

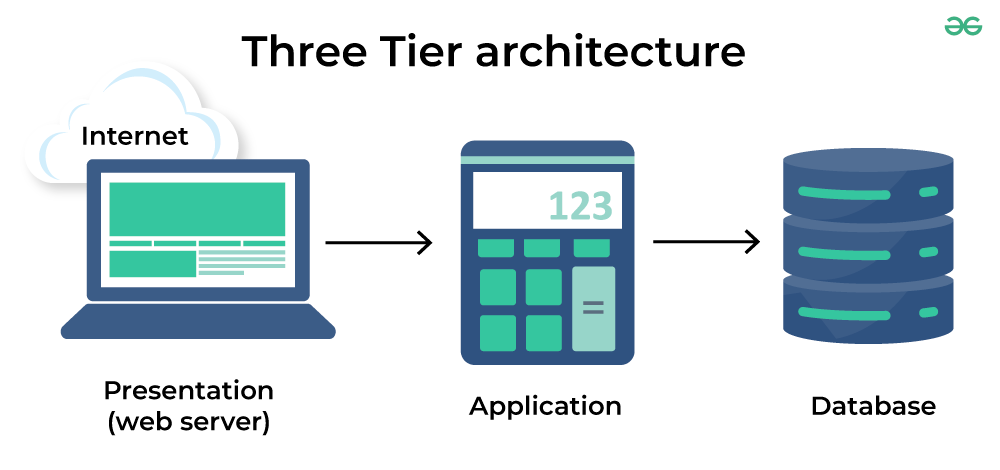
This software manages hardware and system resources or provides a platform for application software.

1. **Windows 10** – Operating system  
   → **System software**

# **Software Architecture**

**Q7.** **Design a basic three-tier software architecture diagram for a web application.**

**A.**



The Three-Tier Client-Server Architecture is a layered approach to building distributed systems, with each tier serving distinct roles. Below is a detailed explanation of each component:

**1. Presentation Tier (Client Tier)**

The user interface and user interaction are under the control of the Presentation Tier. It functions as the application’s front end, where users enter information and see the outcomes.

* **Components:**
  + **User Interface (UI):** Includes web browsers, mobile apps, or desktop applications that users interact with. It displays data and collects user inputs.
  + **User Interaction Logic:** Handles how user inputs are processed and communicated to the Application Tier. This can involve form validation, data formatting, and sending requests to the server.
* **Responsibilities:**
  + Display data from the Application Tier to the user.
  + Collect user inputs and forward them to the Application Tier.
  + Provide a user-friendly interface and manage user interactions.

**2. Application Tier (Business Logic Tier)**

The Application Tier is where the core business logic resides. It processes user requests, performs calculations, enforces business rules, and interacts with the Data Tier to retrieve or store data.

* **Components:**
  + **Business Logic:** Implements the rules and processes specific to the application, such as order processing, authentication, or data validation.
  + **Application Server:** Manages communication between the Presentation and Data Tiers and hosts the business logic. Web servers and application servers such as Apache Tomcat, Microsoft IIS, or JBoss are examples.
* **Responsibilities:**
  + Process and interpret data received from the Presentation Tier.
  + Execute business logic and apply rules.
  + Communicate with the Data Tier to retrieve or store information.
  + Send processed data back to the Presentation Tier for user display.

**3. Data Tier (Database Tier)**

The Data Tier is responsible for data management and storage. It handles all database operations, including data retrieval, updates, and management.

* **Components:**
  + **Database Management System (DBMS):** Software like MySQL, Oracle, or Microsoft SQL Server that manages data storage and retrieval.
  + **Database:** The actual repository where data is stored, organized in tables or other structures.
* **Responsibilities:**
  + Store and manage data securely and efficiently.
  + Handle queries and transactions initiated by the Application Tier.
  + Ensure data integrity, consistency, and availability.
  + Provide backup and recovery mechanisms to protect data.

# **Layers in Software Architecture**

**Q8.** **Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.**

**A. 📚 Case Study: Library Management System**

**🏗️ Overview**

The Library Management System (LMS) is used by a university to manage book checkouts, returns, user registrations, and inventory updates. It follows a **three-tier architecture**:

1. **Presentation Layer** – User interface (web or desktop)
2. **Business Logic Layer** – Application logic and rules
3. **Data Access Layer** – Interacts with the database

**📌 1. Presentation Layer**

**🎯 Purpose:**

Acts as the front-end where users interact with the system.

**🧩 Components:**

* Web pages built with HTML/CSS/JavaScript (or desktop GUI)
* HTTP client requests using REST APIs

**👤 Users:**

* Librarians
* Students
* Administrators

**✅ Functions:**

* Display login form
* Show available books
* Allow book checkout/return
* Display error or success messages

**🔄 Example Workflow:**

* A student logs in → clicks on "Borrow Book" → selects book → frontend sends HTTP POST request to the server.

**🔧 2. Business Logic Layer**

**🎯 Purpose:**

Implements the core rules and workflows of the LMS.

**🧠 Logic Examples:**

* Check if a user has overdue books before allowing a new checkout.
* Enforce borrowing limits (e.g., max 5 books/student).
* Calculate fines for late returns.

**🔄 Example Flow:**

python

def checkout\_book(user\_id, book\_id):

if user\_has\_overdue\_books(user\_id):

raise Exception("Cannot borrow new books until overdue books are returned.")

if not is\_book\_available(book\_id):

raise Exception("Book is currently unavailable.")

create\_loan\_record(user\_id, book\_id)

**✅ Responsibilities:**

* Coordinate between presentation and data layers
* Enforce security and validation
* Handle exceptions and business workflows

**🗄️ 3. Data Access Layer (DAL)**

**🎯 Purpose:**

Provides methods for CRUD (Create, Read, Update, Delete) operations on the database.

**🔧 Technologies:**

* SQL database (e.g., MySQL, PostgreSQL)
* ORM (Object-Relational Mapping) like SQLAlchemy or Hibernate

**🧱 Functions:**

* get\_user\_by\_id(id)
* get\_available\_books()
* insert\_loan\_record(user\_id, book\_id, due\_date)

# **Software Environments**

**Q9.** **Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.**

**A.** There are several types of **software environments** used during the software development lifecycle. Each serves a unique purpose to ensure that software is developed, tested, and deployed in a controlled, efficient, and safe manner.

**🧪 1. Development Environment**

**🔹 Purpose:**

* Where developers **write, build, and test code locally**.

**🔹 Characteristics:**

* Contains compilers, interpreters, text editors, and debugging tools.
* Often runs on personal machines or virtual environments.

**🔹 Example:**

* A developer writing Python code in VS Code using a virtual environment and testing with pytest.

**🔍 2. Testing/QA Environment**

**🔹 Purpose:**

* Used for **automated and manual testing** by QA teams.
* Verifies that the code behaves as expected.

**🔹 Characteristics:**

* Closely mimics production in terms of configuration and data.
* May include staging databases, mock services, and test automation tools.

**🔹 Example:**

* Running end-to-end tests or security tests using Selenium or Postman.

**🧪 3. Staging Environment (Pre-Production)**

**🔹 Purpose:**

* Serves as a **final testing ground before deployment** to production.

**🔹 Characteristics:**

* Almost identical to production.
* Used to simulate real-world usage with production-like data.

**🔹 Example:**

* A release candidate version of a web app is deployed to staging for business review.

**🌐 4. Production Environment**

**🔹 Purpose:**

* Where the **final, user-facing version** of the application runs.

**🔹 Characteristics:**

* Highly secure, stable, and optimized for performance.
* Monitored and backed up regularly.

**🔹 Example:**

* A live e-commerce website used by real customers.

**🧊 5. Sandbox Environment**

**🔹 Purpose:**

* Used for **safe experimentation** or integration testing without affecting live systems.

**🔹 Characteristics:**

* Isolated from other environments.
* Often used by third-party developers or in API testing.

**🔹 Example:**

* A developer uses a payment gateway's sandbox to test API calls.

# **Types of Software**

**Q13.** **Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.**

**A.**

| **Software Name** | **Type of Software** |
| --- | --- |
| **Microsoft Word** | Application Software |
| **Google Chrome** | Application Software |
| **Zoom** | Application Software |
| **Quick Heal** | | Utility Software |
| **Task Manager** | | Utility Software |
| **WinRAR** | Utility Software |
| **Windows** | System Software |
| **Visual Studio Code** | Application Software |
|  |  |

# **Application Software**

**Q15.** **Write a report on the various types of application software and how they improve productivity.**

**A. Introduction**

Application software refers to programs designed to help users perform specific tasks on a computer or other digital device. Unlike system software, which manages hardware and core systems, application software focuses on user-driven functionalities such as creating documents, managing data, communicating, or designing visuals. By automating tasks, streamlining workflows, and enhancing communication, application software plays a critical role in boosting personal and organizational productivity.

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

**1. Word Processing Software**

* **Examples**: Microsoft Word, Google Docs, WPS Office
* **Purpose**: Create, format, and edit text documents.
* **Productivity Benefits**:
  + Speeds up documentation and report writing.
  + Built-in grammar/spell check enhances accuracy.
  + Cloud collaboration allows multiple users to edit in real time.

**2. Spreadsheet Software**

* **Examples**: Microsoft Excel, Google Sheets, LibreOffice Calc
* **Purpose**: Perform calculations, analyse data, and create charts.
* **Productivity Benefits**:
  + Automates complex mathematical functions.
  + Organizes large datasets efficiently.
  + Supports decision-making with visual data tools (charts, pivot tables).

**3. Presentation Software**

* **Examples**: Microsoft PowerPoint, Google Slides, Apple Keynote
* **Purpose**: Create slideshow presentations for educational or business use.
* **Productivity Benefits**:
  + Enhances communication with visual elements.
  + Provides templates to reduce preparation time.
  + Supports teamwork with collaborative editing.

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

* **Examples**: Microsoft Access, MySQL, Oracle DB
* **Purpose**: Store, retrieve, and manage structured data.
* **Productivity Benefits**:
  + Centralizes data access and management.
  + Increases accuracy and reduces data duplication.
  + Improves reporting through query and filtering tools.

**5. Communication Software**

* **Examples**: Zoom, Microsoft Teams, Slack, Gmail
* **Purpose**: Facilitate email, video calls, messaging, and file sharing.
* **Productivity Benefits**:
  + Enables remote work and global collaboration.
  + Reduces response time with instant messaging.
  + Integrates with calendars, task managers, and file systems.

**6. Project Management Software**

* **Examples**: Trello, Asana, Jira, Monday.com
* **Purpose**: Plan, assign, and track tasks within teams.
* **Productivity Benefits**:
  + Visualizes workflow and progress.
  + Enhances team coordination and accountability.
  + Sends reminders and updates to keep projects on track.

**7. Multimedia and Graphic Design Software**

* **Examples**: Adobe Photoshop, Canva, Final Cut Pro, Audacity
* **Purpose**: Edit images, design graphics, create videos and audio.
* **Productivity Benefits**:
  + Speeds up content creation with templates and presets.
  + Enables non-designers to create professional visuals.
  + Enhances branding and presentation of information.

**8. Web Browsers**

* **Examples**: Google Chrome, Mozilla Firefox, Microsoft Edge
* **Purpose**: Access websites and online services.
* **Productivity Benefits**:
  + Provides access to online applications and resources.
  + Supports browser extensions for task automation.
  + Facilitates remote work via cloud-based tools.

# **Software Development Process**

**Q16.** **Create a flowchart representing the Software Development Life Cycle (SDLC)**

**A.**



**Stage-1: Planning and Requirement Analysis**

Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.

**Stage-2: Defining Requirements**

In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.

This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

**Stage-3: Designing Architecture**

SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.

**Stage-4: Developing Product**

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

**Stage-5: Product Testing and Integration**

After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee’s ability to work through learning, usually by changing his attitude and developing his skills and understanding.

**Stage-6: Deployment and Maintenance of Products**

After detailed testing, the conclusive product is released in phases as per the organization’s strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product’s supervision.

# **Software Requirement**

**Q17.** **Write a requirement specification for a simple library management system.**

**A.**

**1. Introduction**

**1.1 Purpose**

This document outlines the functional and non-functional requirements of a Library Management System. The system is intended to manage the daily operations of a small-to-medium-sized library, including book inventory, member management, and borrowing/returning transactions.

**1.2 Scope**

The LMS will allow librarians to manage books and users, track borrow/return status, and generate reports. Library members will be able to search for books, check availability, and request to borrow or return books.

**1.3 Intended Users**

* Librarians (administrators)
* Library Members (users)
* System Administrators

**2. Functional Requirements**

**2.1 User Management**

* Admin can create, edit, and delete member accounts.
* Users can register and log in.
* Role-based access: Admin and Member.

**2.2 Book Management**

* Add new books to the library inventory.
* Edit or remove book records.
* View current availability of books.

**2.3 Borrow/Return Management**

* Members can borrow books (up to a set limit).
* Admins can approve borrow/return transactions.
* Track due dates and overdue books.

**2.4 Search and Filter**

* Users can search for books by title, author, genre, or ISBN.
* Filters include availability and category.

**2.5 Notifications**

* System sends email or dashboard alerts for overdue books and reservation updates.

**2.6 Reporting**

* Generate reports on inventory, borrow history, overdue books, and member activity.

**3. Non-Functional Requirements**

**3.1 Performance**

* The system should support up to 1000 concurrent users.

**3.2 Reliability**

* 99.9% uptime during working hours.

**3.3 Security**

* User authentication with encrypted passwords.
* Admin-only access to sensitive operations.

**3.4 Usability**

* Simple and intuitive user interface for all users.
* Mobile-responsive design.

**3.5 Maintainability**

* Modular architecture to allow easy updates and feature additions.

**4. System Interfaces**

**4.1 User Interface**

* Web-based GUI for users and admins.
* Login, registration, book listing, and dashboard.

**4.2 Database**

* A relational database (e.g., MySQL or PostgreSQL) to store users, books, and transaction records.

**4.3 External Systems**

* Optional integration with email or SMS services for notifications.

**5. Assumptions and Constraints**

* Internet access is required.
* Each member can borrow up to 3 books at a time.
* Books must be returned within 14 days of borrowing.

# **Software Analysis**

**Q18.** **Perform a functional analysis for an online shopping system.**

**A.**

**1. User Management**

**Functions:**

* User Registration
* User Login/Logout
* Profile Management (name, email, address, payment methods)
* Password Recovery

**Users:**

* Customers
* Admins

**2. Product Catalogue Management**

**Functions:**

* Add/Edit/Delete Products (Admin only)
* Categorize Products by type, brand, price range, etc.
* View Product Details (images, description, price, reviews)

**Users:**

* Admin (manage catalogue)
* Customers (view/search products)

**3. Search and Navigation**

**Functions:**

* Search products by name, brand, or category
* Filter/sort by price, rating, popularity
* Product recommendations and related items

**Users:**

* Customers

**4. Shopping Cart**

**Functions:**

* Add items to cart
* Remove items from cart
* Update quantity
* View cart summary (total cost, item count)

**Users:**

* Customers

**5. Order Management**

**Functions:**

* Place an order
* Choose payment and shipping method
* View order confirmation and status
* Cancel or return items

**Users:**

* Customers
* Admin (view/manage all orders)

**6. Payment Processing**

**Functions:**

* Integrate with third-party gateways (e.g., Stripe, PayPal)
* Accept credit/debit cards
* Generate invoices/receipts

**Users:**

* Customers
* Admin (manage payment status)

**7. Shipping and Delivery Tracking**

**Functions:**

* Enter shipping address
* Estimate delivery time
* Track shipment
* Update order delivery status

**Users:**

* Customers
* Admin (update logistics)

**8. Reviews and Ratings**

**Functions:**

* Submit product reviews and ratings
* View average ratings and comments
* Admin moderation of inappropriate content

**Users:**

* Customers
* Admin

**9. Notification System**

**Functions:**

* Email/SMS confirmation for orders, shipping, returns
* Promotional offers
* Low-stock alerts (admin)

**Users:**

* All

**10. Reporting and Analytics**

**Functions:**

* Sales reports
* Product popularity stats
* Customer activity logs

**Users:**

* Admin

# **System Design**

**Q19.** **Design a basic system architecture for a food delivery app.**

**A.** Designing a **basic system architecture** for a food delivery app involves identifying the core components and how they interact. Here's a high-level architecture divided into **four layers**:

**🧱 1. Client Layer (Front-End)**

**Purpose**: Interfaces for user interaction.

* **Mobile Apps (iOS/Android)** for:
  + Customers (browse, order, pay)
  + Delivery drivers (accept deliveries, track routes)
  + Restaurants (manage menu, orders)
* **Web Dashboard** for:
  + Admin (manage users, orders, reports)

**⚙️ 2. Application Layer (Back-End APIs)**

**Purpose**: Business logic and coordination.

* **User Service**: Registration, login, profiles
* **Restaurant Service**: Menu management, restaurant info
* **Order Service**: Cart, checkout, order tracking
* **Delivery Service**: Driver assignment, status updates
* **Payment Service**: Handles integration with payment gateways (e.g., Stripe)
* **Notification Service**: Push notifications, SMS, email
* **Rating/Review Service**

**🗄️ 3. Data Layer**

**Purpose**: Persistent data storage.

* **Relational DB (e.g., PostgreSQL)**:
  + Users, restaurants, orders, payments, ratings
* **NoSQL DB (e.g., MongoDB / Redis)**:
  + Session data, delivery location tracking, cache
* **Cloud Storage (e.g., AWS S3)**:
  + Images (food, profile photos)

**🌐 4. Integration Layer**

**Purpose**: Connect to external services.

* **Payment Gateway** (e.g., Stripe, PayPal)
* **Map/Geolocation API** (e.g., Google Maps)
* **Notification Services** (e.g., Firebase, Twilio)
* **Analytics/Monitoring Tools** (e.g., Mixpanel, Prometheus)

# **Software Testing**

**Q20.** **Develop test cases for a simple calculator program.**

**A.** Here are **test cases for a simple calculator program** that supports basic arithmetic operations: **addition**, **subtraction**, **multiplication**, and **division**.

**✅ Assumptions**

* Inputs are two numbers (integers or floats).
* Operations: +, -, \*, /.
* Handles division by zero.
* Returns correct results with proper data types.

**🧪 Test Case Table**

| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Remarks** |
| --- | --- | --- | --- | --- |
| TC01 | Add two positive integers | add(2, 3) | 5 | Basic addition |
| TC02 | Add two negative integers | add(-2, -3) | -5 | Negative values |
| TC03 | Subtract smaller from larger | subtract(10, 4) | 6 | Basic subtraction |
| TC04 | Subtract larger from smaller | subtract(4, 10) | -6 | Negative result |
| TC05 | Multiply two integers | multiply(3, 5) | 15 | Basic multiplication |
| TC06 | Multiply by zero | multiply(0, 7) | 0 | Edge case |
| TC07 | Divide two integers | divide(10, 2) | 5 | Basic division |
| TC08 | Divide by zero | divide(5, 0) | Error / Exception | Should raise or handle error |
| TC09 | Add floats | add(2.5, 3.2) | 5.7 | Float precision |
| TC10 | Divide float by integer | divide(7.5, 2) | 3.75 | Mixed type operation |
| TC11 | Large number addition | add(1000000, 9999) | 1009999 | Stress test |
| TC12 | Invalid input (string) | add("a", 1) | Error / Type Exception | Input validation |

# **Maintenance**

**Q21.** **Document a real-world case where a software application required critical maintenance.**

**A. 🧠 Case Study: Slack Outage – February 2022**

**🔍 Overview**

Slack, the popular workplace messaging platform, experienced a **major outage** in February 2022, affecting **millions of users globally**. Users were unable to send messages, load channels, or use integrations.

**⚠️ Issue Summary**

* **Symptoms**: Slack failed to load properly; users encountered blank screens or error messages.
* **Duration**: ~3 hours of disruption.
* **Cause**: A **configuration change** in a backend system unexpectedly led to **increased load on the database**.

**🔧 Critical Maintenance Actions Taken**

**1. Rollback Deployment**

* The engineering team **reverted** the recent configuration change to stabilize the platform.

**2. Database Scaling**

* Slack had to **scale its database tier** and add additional read replicas to handle the sudden load surge.

**3. Throttling Traffic**

* Temporarily **throttled some features** (e.g., integrations and notifications) to reduce system strain.

**4. Hotfix Deployment**

* A patched configuration was tested and deployed in controlled stages to prevent another overload.

**5. Monitoring and Postmortem**

* Slack’s team published a **detailed postmortem** and added safeguards to prevent similar issues.

**🧪 Root Cause Analysis**

* A performance-impacting change was deployed without full-load testing.
* A **dependency between services** caused a cascading failure when one service spiked in latency.
* **Insufficient failover and redundancy** in part of the database cluster.

# **DFD (Data Flow Diagram)**

**Q22.** **Create a DFD for a hospital management system.**

**A.** Here's a **Data Flow Diagram (DFD)** for a **Hospital Management System (HMS)** at **Level 1**, which shows the major processes, data stores, external entities, and data flows.

**🏥 Hospital Management System – Level 1 DFD**

**External Entities:**

* **Patient**
* **Doctor**
* **Admin Staff**

**Processes:**

1. **Register Patient**
2. **Schedule Appointment**
3. **Manage Medical Records**
4. **Billing and Payments**
5. **Doctor Consultation**

**Data Stores:**

* **Patient Database**
* **Appointment Schedule**
* **Medical Records**
* **Billing Records**

**🧾 Description of Data Flow**

**1. Register Patient**

* Patient provides personal details → System stores in **Patient Database**

**2. Schedule Appointment**

* Patient requests appointment → System checks doctor availability (from **Appointment Schedule**) → Confirmation sent to patient

**3. Doctor Consultation**

* Doctor retrieves **Patient Data** and updates **Medical Records**

**4. Manage Medical Records**

* Records diagnosis, prescriptions, test results into **Medical Records**

**5. Billing and Payments**

* Admin generates bills from **Medical Records** → Patient makes payment → Stored in **Billing Records**

# **Desktop Application**

**Q23.** **Build a simple desktop calculator application using a GUI library.**

**A. 🧮 Simple Calculator Using Tkinter (Python)**

**python**

import tkinter as tk

def click(event):

button\_text = event.widget["text"]

if button\_text == "=":

try:

result = str(eval(entry.get()))

entry.delete(0, tk.END)

entry.insert(tk.END, result)

except Exception:

entry.delete(0, tk.END)

entry.insert(tk.END, "Error")

elif button\_text == "C":

entry.delete(0, tk.END)

else:

entry.insert(tk.END, button\_text)

# GUI setup

root = tk.Tk()

root.title("Simple Calculator")

root.geometry("300x400")

entry = tk.Entry(root, font="Arial 20")

entry.pack(fill=tk.BOTH, ipadx=8, pady=10)

# Button labels

buttons = [

["7", "8", "9", "/"],

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

["1", "2", "3", "-"],

["0", "C", "=", "+"]

]

# Create buttons in a grid

for row in buttons:

frame = tk.Frame(root)

frame.pack(expand=True, fill=tk.BOTH)

for label in row:

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

btn.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)

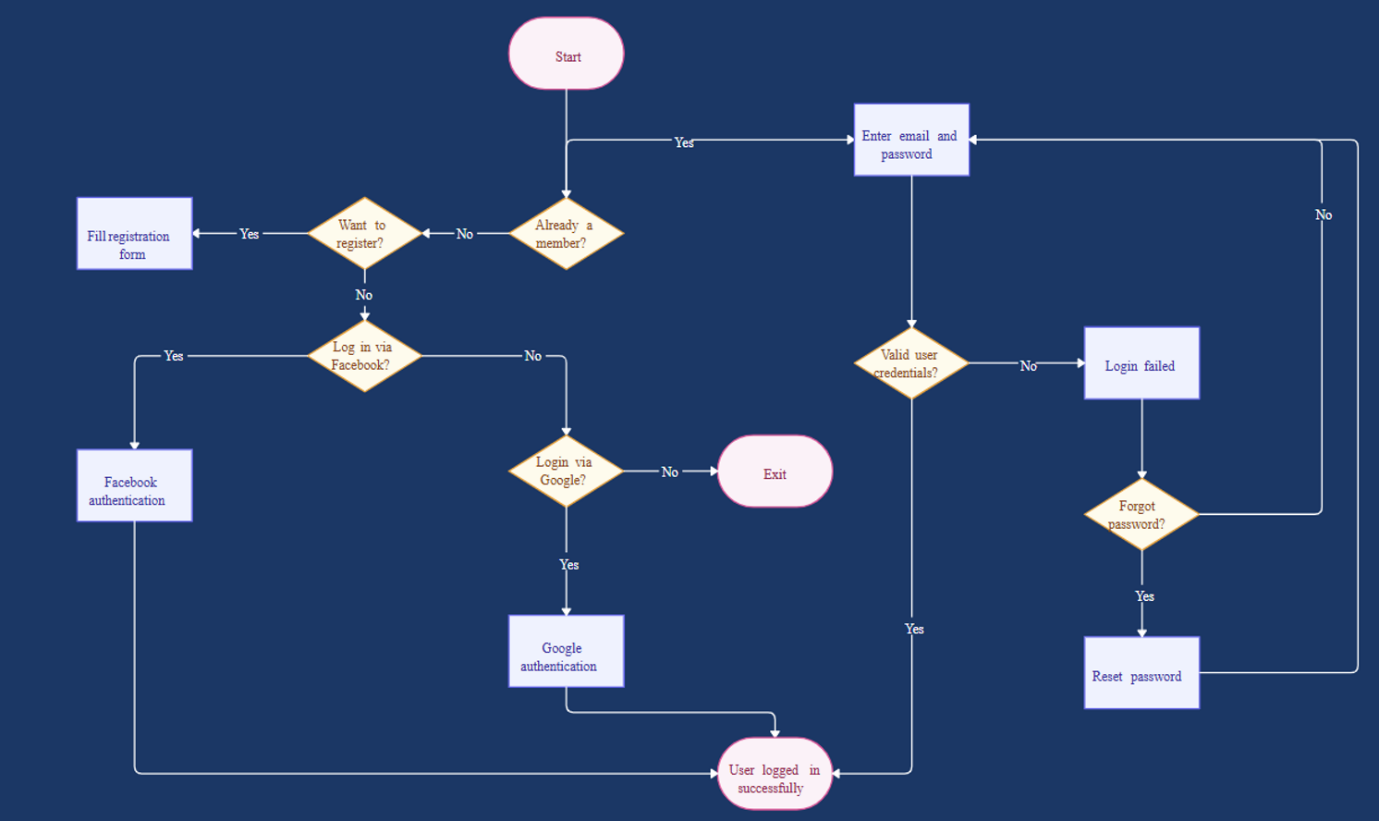
btn.bind("<Button-1>", click)

root.mainloop()

# **Desktop Application**

**Q24.** **Draw a flowchart representing the logic of a basic online registration system.**

**A.**



END