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

Ans:- # Hello World in Python

print("Hello, World!")

// Hello World in C

#include <stdio.h>

main()

{

printf("Hello, World!");

}

**Comparison of Structure and Syntax:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Python** | **C** |
| Simplicity | Extremely simple (one line) | Needs function and library inclusion |
| Boilerplate Code | No boilerplate | Requires header and main function |
| Syntax Style | Very clean and readable | Structured and manual (more setup) |
| Compilation | Directly interpreted | Needs to be compiled first (gcc file.c) |
| Printing | print("text") | printf("text\n"); |

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

Ans:- When a client sends data to a server , the data moves through several steps:

1. **Client** sends a request (like opening a website).
2. The data is broken into **packets**.
3. It goes through the **local network** (like Wi-Fi or Ethernet).
4. Then it travels through your **Internet Service Provider (ISP)**.
5. The packets cross **various routers** across the **Internet backbone**.
6. Finally, they reach the **Server**, which responds back the same way.

Daigram of that flow:

[Client Device]

↓

[Local Network Router/Wi-Fi]

↓

[Internet Service Provider (ISP)]

↓

[Internet Backbone (Multiple Routers and Networks)]

↓

[Destination Server]

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

Ans:- Client Server

| |

| ---- (HTTP Request: GET) ----> |

| |

|<---- (HTTP Response: 200 OK) -|

| |+

* The Client sends an HTTP request (like "GET /something").
* The Server receives the request and sends back an HTTP response (like "200 OK" + some data).

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

Ans**:- List of Internet connection types:**

1. **DSL (Digital Subscriber Line)**
2. **Cable Internet**
3. **Fiber-Optic Internet**
4. **Satellite Internet**
5. **Fixed Wireless Internet**
6. **Dial-Up Internet**
7. **Mobile Internet (4G/5G)**

**Pros and cons of each:**

**1. DSL (Digital Subscriber Line)**

* Pros:
  + Widely available.
  + Affordable for most users.
* Cons:
  + Slower speeds compared to cable or fiber.
  + Performance depends on distance from the service provider.

**2. Cable Internet**

* Pros:
  + Faster than DSL.
  + Good for activities like streaming and gaming.
* Cons:
  + Shared with neighbors — speeds may drop during peak hours.

**3. Fiber-Optic Internet**

* Pros:
  + Extremely fast (up to 1 Gbps or more).
  + Very reliable and stable.
* Cons:
  + Limited availability in some areas.
  + Installation can be expensive.

**4. Satellite Internet**

* Pros:
  + Available almost everywhere (good for remote or rural areas).
* Cons:
  + High latency (causes lag).
  + Can be affected by bad weather.
  + Often has data limits (caps).

**5. Fixed Wireless Internet**

* Pros:
  + Quick setup.
  + Good for places without cable/fiber access.
* Cons:
  + Needs a clear line of sight to a tower.
  + Weather can affect the signal strength.

**6. Dial-Up Internet**

* Pros:
  + Very cheap.
  + Only needs a basic telephone line.
* Cons:
  + Extremely slow (almost outdated now).
  + Blocks the phone line during use.

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

* Pros:
  + Portable and mobile — you can access it almost anywhere.
  + High speed with 5G, good for streaming and gaming.
* Cons:
  + Coverage depends on your area.
  + Data plans can be expensive.

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

Ans:- **1) Simulate an HTTP Request using curl**

**example:** Fetch a webpage from a server.

bash

curl http://example.com

* **What happens?**
  + Sends an **HTTP GET** request to http://example.com.
  + Prints the HTML content of the page.

**Check only the headers:**

Bash

curl -I http://example.com

**Send a POST request:**

Bash

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

2) Simulate an **FTP Request** using ftp or curl

🡪 **ftp ftp.example.com**

i) It will ask for **Username** and **Password**

ii) After login, you can use commands like:

* ls — list files
* get filename — download file
* put filename — upload file
* bye — exit FTP session
* **Download a file via FTP:**

bash

curl ftp://ftp.example.com/file.txt --user username:password

* **Upload a file via FTP:**

bash

curl -T localfile.txt ftp://ftp.example.com/ --user username:password

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

Ans:- **Three Common Application Security Vulnerabilities:**

**1. SQL Injection**

What it is:

* When an attacker inserts malicious SQL code into an application's input field (like a login form) to manipulate or access the database.

Possible Solutions:

* Use Prepared Statements (Parameterized Queries) — so SQL and user input stay separate.
* Validate and sanitize all user inputs — reject suspicious data.
* Use ORM frameworks (like Django ORM, Hibernate) which manage database queries safely.

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

**What it is:**

* When an attacker injects malicious scripts (like JavaScript) into a webpage that is then viewed by other users.
* Example: Attacker posts a comment like:

html

<script>alert('Hacked!');</script>

Possible Solutions:

* Escape all output — any data from users should be treated as dangerous and escaped properly before displaying it.
* Use Content Security Policy (CSP) — a browser feature that limits which scripts can run.
* Input validation — restrict what users are allowed to submit.

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

What it is:

* When an attacker tricks a logged-in user into unknowingly sending a request to a web application (like changing a password or making a purchase).
* Example: Clicking on a malicious link that sends a hidden request.

Possible Solutions:

* Use CSRF tokens — a secret random value added to forms that the server checks.
* SameSite cookies — configure cookies to not be sent along with cross-site requests.
* Require re-authentication for sensitive actions.

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

Ans:- Here’s a simple classification of 5 applications I might use daily, classifying them as either **system software** or **application software**:

**1. Operating System (e.g., Windows)**

* Type: System Software
* Why: It provides the fundamental interface between hardware and the software applications you use.

**2. Web Browser (e.g., Chrome)**

* Type: Application Software
* Why: Used to browse the internet and run web applications, not part of the system but built on top of the OS.

**3. Microsoft Word or Google Docs**

* Type: Application Software
* Why: These are used for creating and editing documents, and are designed to perform specific tasks for the user.

**4. Antivirus Software (e.g., McAfee, Norton)**

* Type: System Software
* Why: It operates at a system level to monitor and protect the computer from malware, helping secure the system.

**5. Media Player (e.g., VLC, Windows Media Player)**

* Type: Application Software
* Why: Used for playing audio and video files; designed for specific user tasks like media consumption.

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

Ans:-

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

Ans:- **Case Study: Layered Architecture of an Online Clothing E-Commerce System**

**Introduction**

In modern software engineering, multi-layered architecture helps separate concerns and improve system scalability, maintainability, and testability. This case study analyses the functionalities of the Presentation Layer, Business Logic Layer (BLL), and Data Access Layer (DAL) of an Online Clothing E-Commerce System.

**System Overview**

The system allows customers to:

* Browse clothing products
* Add products to a shopping cart
* Manage user accounts
* Place orders and make payments

The system uses a classic **three-tier architecture**:

* **Presentation Layer** (Frontend)
* **Business Logic Layer** (Application Server)
* **Data Access Layer** (Database Communication)

**1. Presentation Layer Functionality**

**Purpose:**

The Presentation Layer is responsible for displaying information to the user and capturing user input.

**Key Functions:**

* **User Interface Rendering**: Displays product listings, shopping cart contents, and checkout forms through HTML/CSS/JavaScript or mobile app screens.
* **Input Handling**: Captures user actions such as search queries, add-to-cart operations, and payment details.
* **Data Binding**: Binds the user input fields with application objects (e.g., product objects).
* **Communication with BLL**: Sends HTTP requests to the business logic layer to process user actions.

**Example:**

* A customer clicks "Add to Cart" on a product page.
* The page sends a POST request to the server API (/api/cart/add) with product ID and quantity.
* Displays confirmation or error messages based on the server response.

**2. Business Logic Layer (BLL) Functionality**

**Purpose:**

The Business Logic Layer contains the core application logic — it processes user requests, enforces rules, and controls the flow of data between the UI and the database.

**Key Functions:**

* **Input Validation**: Ensures data is correct before passing it to the DAL (e.g., verifying that a product exists before adding it to the cart).
* **Business Rules Enforcement**: Enforces rules such as inventory checks (cannot order more items than are in stock) or user authentication requirements.
* **Service Coordination**: Coordinates calls between the Presentation Layer and DAL, possibly integrating multiple DAL operations into a cohesive service.
* **Session Management**: Maintains user sessions, cart sessions, and handles secure transactions.

**Example:**

* The BLL receives a request to add a product to the cart.
* It validates if the user session is active.
* It checks if the product ID is valid and if sufficient stock exists.
* If valid, it updates the cart model and responds back to the Presentation Layer.

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

**Purpose:**

The Data Access Layer directly interacts with the database. It abstracts the complexity of data persistence and retrieval.

**Key Functions:**

* **Database Connectivity**: Manages connections to relational databases (e.g., PostgreSQL, MySQL) or NoSQL stores (e.g., MongoDB).
* **CRUD Operations**: Performs Create, Read, Update, Delete operations.
* **Data Mapping**: Maps database rows to application models (e.g., ORM tools like Django ORM or Entity Framework).
* **Data Integrity Management**: Ensures transactions are atomic and rollback on failure.

**Example:**

* BLL requests the DAL to fetch a product by ID.
* DAL runs SQL: SELECT \* FROM Products WHERE id = 123.
* DAL maps the database row to a Product object and returns it to the BLL.

**Interactions Between Layers**

[User Interaction]

↓

[Presentation Layer]

↓ API call

[Business Logic Layer]

↓ DB queries

[Data Access Layer]

↔ Database

Example Flow for "Add to Cart":

1. User clicks "Add to Cart" → Presentation Layer sends request.
2. BLL validates request → checks product availability.
3. DAL fetches product data → updates cart data in database.
4. BLL sends response → Presentation Layer updates UI.

**Benefits of Layered Approach**

* **Maintainability**: Changes to one layer (e.g., switching databases) do not impact others.
* **Scalability**: Can scale layers independently (e.g., multiple application servers).
* **Security**: Sensitive logic stays on the server side in BLL and DAL.
* **Reusability**: Business logic can be reused across different frontends (e.g., web, mobile).

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

Ans:- **Different Types of Software Environments**

In software development, different environments are used to ensure that code is correctly written, tested, and deployed without causing issues for users.

**1. Development Environment (DEV)**

* **Purpose**: For developers to build and write code.
* **Characteristics**:
  + Frequently updated.
  + Debugging tools enabled.
  + Minimal security restrictions.
  + Local databases or mocked services.
* **Example**: A developer's laptop running a Django project with a local SQLite database.

**2. Testing Environment (TEST)**

* **Purpose**: To test the software after development but before it is moved to production.
* **Characteristics**:
  + Contains real or realistic test data.
  + Might include automated and manual testing (Unit, Integration, System testing).
  + Closer to production configuration.
* **Example**: A server where QA engineers run Selenium scripts to check for bugs.

**3. Production Environment (PROD)**

* **Purpose**: Live environment where end users interact with the software.
* **Characteristics**:
  + Highly secure, monitored, and maintained.
  + High performance required.
  + Critical — any bugs can affect users.
* **Example**: A real website (like Amazon.com) that users buy from

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

Ans:-

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

Ans:-

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

Ans:-

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

Ans:- **Your Software List and Classification**

|  |  |  |
| --- | --- | --- |
| **Category** | **Software** | **Purpose** |
| **System Software** | Windows 11 | Desktop operating system to manage hardware and resources. |
|  | Android OS | Mobile operating system for smartphones and tablets. |
|  | Device Drivers (e.g., Printer Driver, GPU Driver) | Interface between the OS and hardware components. |

|  |  |  |
| --- | --- | --- |
| **Application Software** | **Software** | **Purpose** |
|  | Microsoft Word | Document editing and word processing. |
|  | Chrome | Web browsing. |
|  | Microsoft Edge | Web browsing. |
|  | WhatsApp | Messaging and communication. |
|  | Telegram | Messaging and secure communication. |
|  | Zoom | Video conferencing and online meetings. |
|  | Spotify | Music streaming and playback. |
|  | Canva | Graphic design and content creation. |
|  | Dev++ (Dev-C++) | C/C++ programming IDE (Integrated Development Environment). |
|  | Instagram | Social media platform for sharing images and videos. |

|  |  |  |
| --- | --- | --- |
| **Utility Software** | **Software** | **Purpose** |
|  | WinRAR / 7-Zip | File compression and extraction. |
|  | CCleaner | System cleaning and optimization. |
|  | Windows Defender | Antivirus protection built into Windows. |
|  | Avast Antivirus | Third-party virus scanning and protection. |
|  | Backup and Restore (Windows) | Data backup and recovery utility (no macOS tools included). |
|  | Disk Cleanup Tool (Windows) | Frees up space on the hard drive by deleting unnecessary files. |
|  | Partition Wizard / Disk Management Tool | Disk partitioning and management. |

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

Ans:-

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

Ans:- **Types of Application Software**

**1. Word Processing Software**

* **Examples**: Microsoft Word, Google Docs, WPS Office
* **Purpose**: Used to create, edit, format, and print text documents.
* **Productivity Improvement**:
  + Speeds up the writing process with tools like templates, grammar checking, and auto-correct.
  + Enables collaboration through features like real-time editing and comments.

**2. Spreadsheet Software**

* **Examples**: Microsoft Excel, Google Sheets
* **Purpose**: Organizes, calculates, and analyses data using tables and mathematical formulas.
* **Productivity Improvement**:
  + Automates complex calculations and data analysis.
  + Helps in financial planning, reporting, and decision-making through graphs and pivot tables.

**3. Presentation Software**

* **Examples**: Microsoft PowerPoint, Google Slides, Canva
* **Purpose**: Used to create visual slideshows for meetings, classes, and marketing.
* **Productivity Improvement**:
  + Makes it easy to communicate ideas clearly and attractively.
  + Supports teamwork through shared presentation building and commenting.

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

* **Examples**: MySQL, Microsoft Access, Oracle
* **Purpose**: Stores, retrieves, and manages large volumes of data efficiently.
* **Productivity Improvement**:
  + Organizes critical information, reducing the time needed to retrieve or update data.
  + Enhances data security and integrity.

**5. Communication Software**

* **Examples**: WhatsApp, Zoom, Microsoft Teams, Gmail
* **Purpose**: Facilitates instant messaging, email, video conferencing, and collaboration.
* **Productivity Improvement**:
  + Saves time through faster communication and file sharing.
  + Supports remote work and global collaboration.

6. **Types of Application Software**

**1. Word Processing Software**

* **Examples**: Microsoft Word, Google Docs, WPS Office
* **Purpose**: Used to create, edit, format, and print text documents.
* **Productivity Improvement**:
  + Speeds up the writing process with tools like templates, grammar checking, and auto-correct.
  + Enables collaboration through features like real-time editing and comments.

**2. Spreadsheet Software**

* **Examples**: Microsoft Excel, Google Sheets
* **Purpose**: Organizes, calculates, and analyzes data using tables and mathematical formulas.
* **Productivity Improvement**:
  + Automates complex calculations and data analysis.
  + Helps in financial planning, reporting, and decision-making through graphs and pivot tables.

**3. Presentation Software**

* **Examples**: Microsoft PowerPoint, Google Slides, Canva
* **Purpose**: Used to create visual slideshows for meetings, classes, and marketing.
* **Productivity Improvement**:
  + Makes it easy to communicate ideas clearly and attractively.
  + Supports teamwork through shared presentation building and commenting.

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

* **Examples**: MySQL, Microsoft Access, Oracle
* **Purpose**: Stores, retrieves, and manages large volumes of data efficiently.
* **Productivity Improvement**:
  + Organizes critical information, reducing the time needed to retrieve or update data.
  + Enhances data security and integrity.

**5. Communication Software**

* **Examples**: WhatsApp, Zoom, Microsoft Teams, Gmail
* **Purpose**: Facilitates instant messaging, email, video conferencing, and collaboration.
* **Productivity Improvement**:
  + Saves time through faster communication and file sharing.
  + Supports remote work and global collaboration.

6. **Web Browsers**

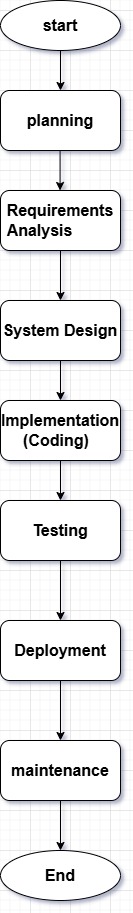
* **Examples**: Google Chrome, Mozilla Firefox, Microsoft Edge
* **Purpose**: Accesses information on the internet.
* **Productivity Improvement**:
  + Provides quick access to resources, research material, and cloud services.
  + Supports extensions and plugins that enhance workflow (e.g., Grammarly, Adblocker).

**How Application Software Improves Productivity Overall**

* **Automation**: Replaces repetitive manual tasks with fast, automated processes.
* **Collaboration**: Enables multiple users to work on the same document or project from different locations.
* **Accuracy**: Reduces human error through built-in checks, templates, and formula-based calculations.
* **Accessibility**: Allows remote access to data and services anytime, improving flexibility.
* **Organization**: Helps users manage projects, schedules, and communications efficiently.
* **Cost-Effectiveness**: Saves time and money by improving task execution and reducing the need for physical resources (e.g., paper).

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

Ans:-



# 16. Write a requirement specification for a simple library management system.

Ans:- **Project Title: Library Management System**

**1. Introduction**

The Library Management System is a software application designed to handle the day-to-day operations of a library. It allows librarians to manage books, members, and borrowing/returning activities efficiently. The system aims to automate library operations to save time, reduce errors, and improve service quality.

**2. Objectives**

* Enable librarians to add, update, delete, and search books.
* Manage member registrations and their borrowing records.
* Track issued and returned books accurately.
* Generate basic reports (e.g., borrowed books, overdue books).

**3. Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| FR1 | Librarians should be able to add new books with details like title, author, ISBN, and quantity. |
| FR2 | Librarians should be able to search, update, or delete book records. |
| FR3 | Members should be registered with personal details (Name, ID, Contact Info). |
| FR4 | Librarians should issue a book to a member and record the issue date and due date. |
| FR5 | Librarians should accept returned books and update records accordingly. |
| FR6 | The system should prevent issuing books if none are available. |
| FR7 | Generate a list of books currently issued and highlight overdue books. |

**4. Non-Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| NFR1 | The system should be user-friendly and easy to navigate. |
| NFR2 | The database must ensure data security and prevent unauthorized access. |
| NFR3 | System response time for any operation (like search or book issue) should be less than 2 seconds. |
| NFR4 | Backup of the database should occur automatically every day. |

**5. System Requirements**

|  |  |
| --- | --- |
| **Component** | **Specification** |
| Hardware | Desktop/Server with 4GB RAM, 500GB HDD, 2.0 GHz processor or higher. |
| Software | Windows/Linux OS, MySQL database, Python/PHP/Java (for backend), HTML/CSS/JavaScript (for frontend). |
| Network | LAN/Wi-Fi for multi-user access (optional). |

**6. User Roles**

|  |  |
| --- | --- |
| **Role** | **Permissions** |
| Librarian | Full access to manage books, members, issue/return books, and view reports. |
| Member | (Optional for advanced systems) Can view available books online and request a hold. |

**7. Assumptions**

* Only librarians will manage the system; members won't directly log into the system .
* Each member can borrow up to 3 books at a time.
* Books must be returned within 14 days unless renewed.

**8. Constraints**

* Must operate offline if internet connectivity is unavailable.
* Should support up to 10,000 book records without performance degradation.

**9. Glossary**

* **ISBN**: International Standard Book Number.
* **Overdue**: A book that has not been returned by its due date.
* **Issue Date**: The date when a book is borrowed.

# 17. : Perform a functional analysis for an online shopping system.

Ans:- **1. Introduction**

The Online Shopping System allows users to browse products, add them to a cart, place orders, and make payments. Administrators manage products, orders, and customer details through a backend system.

**2. Core Functional Areas**

|  |  |
| --- | --- |
| **Functional Area** | **Description** |
| User Management | Registration, login, profile update, and password recovery. |
| Product Management | Add, edit, remove, and categorize products (for admin users). |
| Product Catalog Browsing | Users can browse products by category, search, and view product details. |
| Shopping Cart Management | Users can add, update, or remove products from the shopping cart. |
| Order Management | Users can place orders; admins can view and manage all orders. |
| Payment Processing | Secure checkout, support for multiple payment methods (credit/debit cards, wallets). |
| Inventory Management | Automatically update stock levels after each purchase. |
| Shipping and Delivery Tracking | Users can select shipping options and track their orders. |
| Customer Support | Users can contact support or check FAQs/help sections. |
| Review and Rating System | Customers can rate and review products after purchase. |

**3. Key Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| FR1 | Users must be able to register and log into the system securely. |
| FR2 | Users should be able to browse and search for products easily. |
| FR3 | The system must allow users to add products to the cart and modify it before purchase. |
| FR4 | The system should process payments securely and update order status. |
| FR5 | Admins must manage (add/update/delete) product listings. |
| FR6 | Inventory should automatically decrease when a product is purchased. |
| FR7 | The system should notify users via email/SMS upon successful order placement. |
| FR8 | Users should be able to view their order history and current order status. |
| FR9 | Customers should have the ability to leave reviews for purchased products. |
| FR10 | The platform should provide a customer service contact form or chatbot. |

**4. Actors and Their Functions**

|  |  |
| --- | --- |
| **Actor** | **Functions** |
| **Guest User** | Browse products, register, and login. |
| **Registered User** | Add to cart, checkout, payment, view orders, review products. |
| **Administrator** | Manage products, manage users, view and process orders, manage inventory. |

# 18. Design a basic system architecture for a food delivery app.

Ans:- **1. Architecture Overview**

The system is typically built using a 3-tier architecture:

**1. Presentation Layer (Frontend)**

* **User App** (Customers)
* **Restaurant App** (Partners)
* **Delivery Partner App**
* **Admin Web Dashboard**

**2. Application Layer (Backend/Business Logic)**

* Handles core logic, APIs, routing, authentication, etc.
* Frameworks: Django, Node.js, Laravel, etc.

**3. Data Layer (Database & Storage)**

* Relational DB (e.g., MySQL) for users, orders, menus
* NoSQL DB (e.g., MongoDB) for logs, notifications
* Cloud storage for media (images of food, receipts)

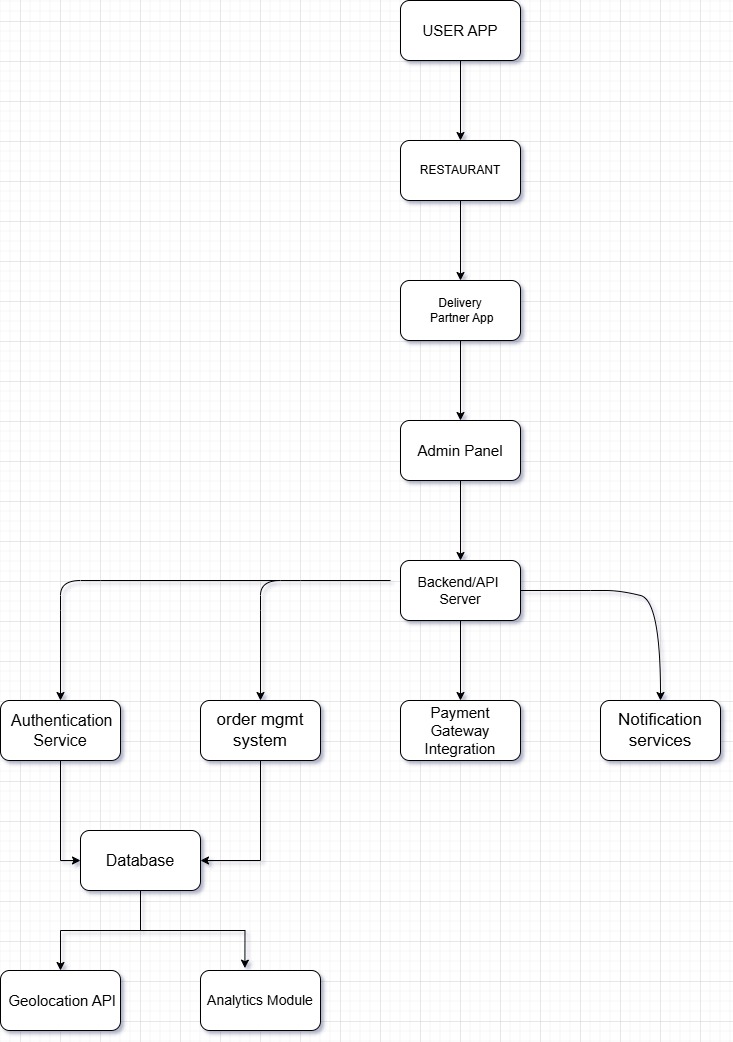
**2. Key System Components**

|  |  |
| --- | --- |
| **Component** | **Description** |
| **User Interface (App/Web)** | Enables customers to browse restaurants, place orders, track delivery. |
| **Restaurant Panel** | Allows restaurants to manage menus, prices, orders. |
| **Delivery Partner App** | Assigns delivery tasks, tracks status, and route optimization. |
| **Admin Panel** | Manages users, restaurants, commissions, reviews, and analytics. |
| **Authentication Service** | Manages user login, token generation, password resets. |
| **Order Management System** | Handles order creation, updates, cancellations, and status tracking. |
| **Payment Gateway Integration** | Enables secure payment processing via Razorpay, Stripe, etc. |
| **Real-Time Notification Service** | Sends order updates via push, SMS, or email. |
| **Geolocation & Map API** | Tracks delivery drivers, estimates arrival times, and shows routes. |
| **Analytics & Reporting Module** | Provides insights for admin and restaurants. |

**4. Technologies (Optional)**

|  |  |
| --- | --- |
| **Layer** | **Technologies** |
| Frontend | Flutter, React Native, HTML/CSS |
| Backend/API | Django, Express.js, Laravel |
| Database | PostgreSQL, Firebase, MongoDB |
| Maps/Tracking | Google Maps API, Mapbox |
| Payment Gateway | Stripe, Razorpay, PayPal |
| Hosting | AWS, Firebase, Heroku |

* **Sample Architecture Diagram**



# 19. Develop test cases for a simple calculator program.

Ans:-**Test Cases for Simple Calculator**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Remarks** |
| TC001 | Add two positive numbers | 5 + 3 | 8 | Basic addition |
| TC002 | Add negative and positive number | -2 + 4 | 2 | Check with negative input |
| TC003 | Subtract two numbers | 10 - 7 | 3 | Basic subtraction |
| TC004 | Subtract with negative result | 3 - 10 | -7 | Result is negative |
| TC005 | Multiply two numbers | 6 \* 4 | 24 | Basic multiplication |
| TC006 | Multiply with zero | 7 \* 0 | 0 | Zero multiplication |
| TC007 | Divide two numbers | 8 / 2 | 4 | Basic division |
| TC008 | Divide by one | 9 / 1 | 9 | Identity division |
| TC009 | Divide by zero | 5 / 0 | Error/Exception | Should handle gracefully |
| TC010 | Add decimal numbers | 2.5 + 3.1 | 5.6 | Floating-point addition |
| TC011 | Subtract decimal numbers | 5.0 - 1.2 | 3.8 | Decimal subtraction |
| TC012 | Large number addition | 999999 + 1 | 1000000 | Boundary test |
| TC013 | Invalid input (letters) | "a" + 2 | Error/Invalid Input | Input validation |

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

Ans:- **Case Study: Critical Maintenance of Facebook iOS App (May 2020)**

**1. Introduction**

In May 2020, thousands of iOS users reported that popular apps such as Facebook, TikTok, Spotify, Pinterest, and others were crashing immediately after launch. The root cause was traced back to a faulty update in Facebook's SDK (Software Development Kit) used for login and analytics.

**2. Problem Overview**

* **Impact**: Apps using the Facebook SDK were affected globally.
* **Cause**: A configuration update pushed by Facebook servers caused the SDK to crash when initializing in the client apps—even without code changes in those apps.
* **Nature of Issue**: The issue was server-side but impacted client-side applications, requiring emergency maintenance.

**3. Critical Maintenance Actions**

|  |  |
| --- | --- |
| **Action** | **Description** |
| **Issue Identification** | Developers identified the crash as originating from Facebook SDK using crash logs. |
| **User Impact Mitigation** | Some developers temporarily disabled Facebook-related features or login options. |
| **Facebook Response** | Facebook rolled back the problematic server-side change that triggered the SDK error. |
| **Recovery Monitoring** | Continuous monitoring was done to ensure the fix worked across all affected apps. |
| **Post-Mortem Analysis** | Facebook and developers reviewed SDK integration methods and how to decouple critical app features from third-party SDKs. |

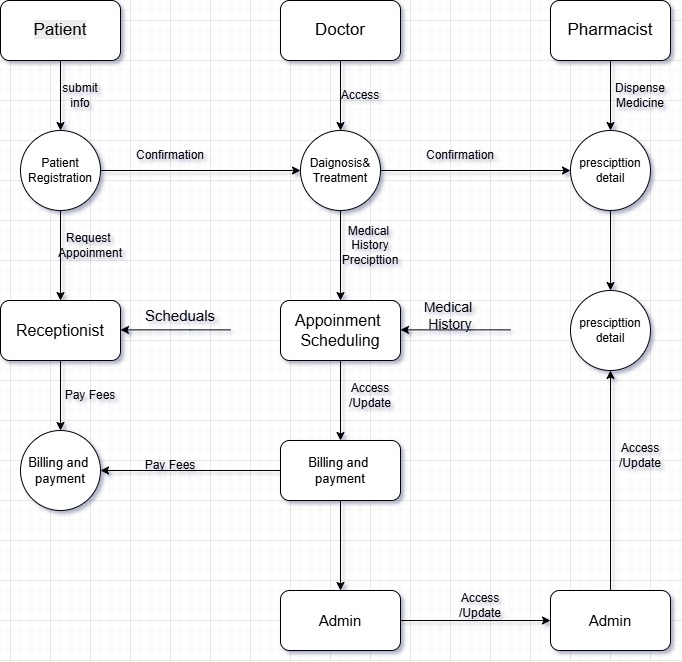
**4. Lessons Learned**

* Server-side changes can break client apps even if the apps haven't been updated.
* Apps should fail gracefully when SDKs or APIs malfunction.
* Decoupling third-party services from core app functionality is crucial.
* Monitoring tools and crash analytics (e.g., Firebase, Crashlytics) are essential for rapid response.

**5. Outcome**

* Apps were restored within hours after Facebook reverted the change.
* This incident led to stronger caution around third-party SDK dependencies and better maintenance protocols in many mobile development teams.

# 21. : Create a DFD for a hospital management system.

Ans:-

# 22. Build a simple desktop calculator application using a GUI library.

Ans:

#include <gtk/gtk.h>

#include <stdlib.h>

GtkWidget \*entry;

void on\_button\_clicked(GtkWidget \*widget, gpointer data) {

const gchar \*label = gtk\_button\_get\_label(GTK\_BUTTON(widget));

const gchar \*current = gtk\_entry\_get\_text(GTK\_ENTRY(entry));

if (g\_strcmp0(label, "=") == 0) {

// Simple evaluation using system call (not safe for complex or real use)

char command[256];

char result[128];

FILE \*fp;

snprintf(command, sizeof(command), "echo 'scale=2; %s' | bc", current);

fp = popen(command, "r");

if (fp == NULL) return;

if (fgets(result, sizeof(result), fp) != NULL) {

gtk\_entry\_set\_text(GTK\_ENTRY(entry), result);

}

pclose(fp);

} else if (g\_strcmp0(label, "C") == 0) {

gtk\_entry\_set\_text(GTK\_ENTRY(entry), "");

} else {

gchar \*new\_text = g\_strconcat(current, label, NULL);

gtk\_entry\_set\_text(GTK\_ENTRY(entry), new\_text);

g\_free(new\_text);

}

}

int main(int argc, char \*argv[]) {

gtk\_init(&argc, &argv);

GtkWidget \*window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_title(GTK\_WINDOW(window), "Calculator");

gtk\_window\_set\_default\_size(GTK\_WINDOW(window), 250, 300);

g\_signal\_connect(window, "destroy", G\_CALLBACK(gtk\_main\_quit), NULL);

GtkWidget \*vbox = gtk\_box\_new(GTK\_ORIENTATION\_VERTICAL, 2);

entry = gtk\_entry\_new();

gtk\_box\_pack\_start(GTK\_BOX(vbox), entry, FALSE, FALSE, 2);

const gchar \*buttons[4][4] = {

{"7", "8", "9", "/"},

{"4", "5", "6", "\*"},

{"1", "2", "3", "-"},

{"C", "0", "=", "+"}

};

for (int i = 0; i < 4; i++) {

GtkWidget \*hbox = gtk\_box\_new(GTK\_ORIENTATION\_HORIZONTAL, 2);

for (int j = 0; j < 4; j++) {

GtkWidget \*button = gtk\_button\_new\_with\_label(buttons[i][j]);

g\_signal\_connect(button, "clicked", G\_CALLBACK(on\_button\_clicked), NULL);

gtk\_box\_pack\_start(GTK\_BOX(hbox), button, TRUE, TRUE, 2);

}

gtk\_box\_pack\_start(GTK\_BOX(vbox), hbox, TRUE, TRUE, 2);

}

gtk\_container\_add(GTK\_CONTAINER(window), vbox);

gtk\_widget\_show\_all(window);

gtk\_main();

return 0;

}

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

Ans:-