❖Module 1 – Overview of IT Industry

(LAB EXERCISE)

LAB EXERCISE 1:

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

Ans:

Here's an example in **Python** and **C** with a short comparison:

1. Python:

```
print("Hello World")
```

2. C:

```
#include <stdio.h>
#include<conio.h>

Void  main()
 {
      printf("Hello World");
}
```

Comparison:

Feature	Python	C
Length	Very short (1 line)	Longer (needs multiple lines)
Setup	No need for special setup or headers	Needs header file (#include <stdio.h>)
Main function	Not required	Must have main() function
End of statement	No semicolon needed	Must end with ;
Readability	More like plain English	More structured for the computer

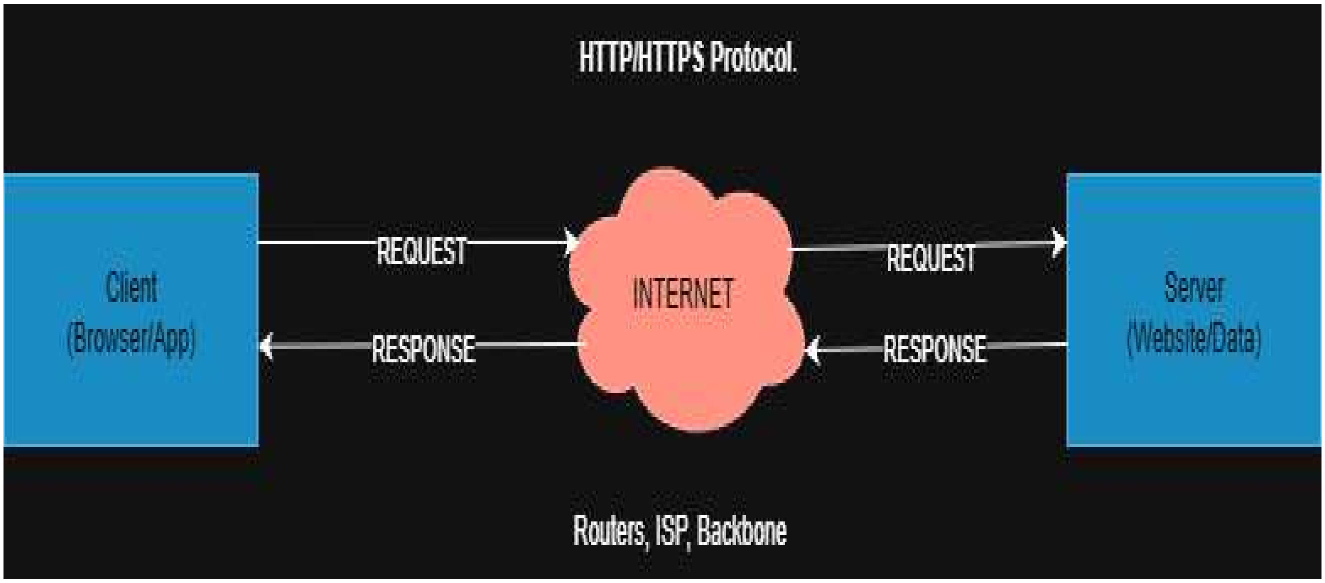
LAB EXERCISE 4:

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

Ans:

Client-Server Communication: How Data is Transmitted Over the Internet.

1. **Client Initiates a Request:**
   - The **client** (e.g., a web browser or mobile app) sends a request to access data or services.
   - This request is usually made using **HTTP or HTTPS** protocols.
2. **Data Travels Through the Internet:**
   - The request is sent through various **routers**, **Internet Service Providers (ISP)**, and the **internet backbone**.
   - Each hop routes the data closer to the **server**.
3. **Server Processes the Request:**
   - The **server** (hosting the website or data) receives the request.
   - The server processes it, accesses its database or application logic, and prepares the **response**.
4. **Response Sent Back to the Client:**
   - The server sends the requested data (HTML, JSON, images, etc.) back to the client.
   - The response follows the reverse path through routers, ISP, and the internet backbone.
5. **Client Displays Data:**
   - The client processes the response and displays the content to the user.



LAB EXERCISE 5:

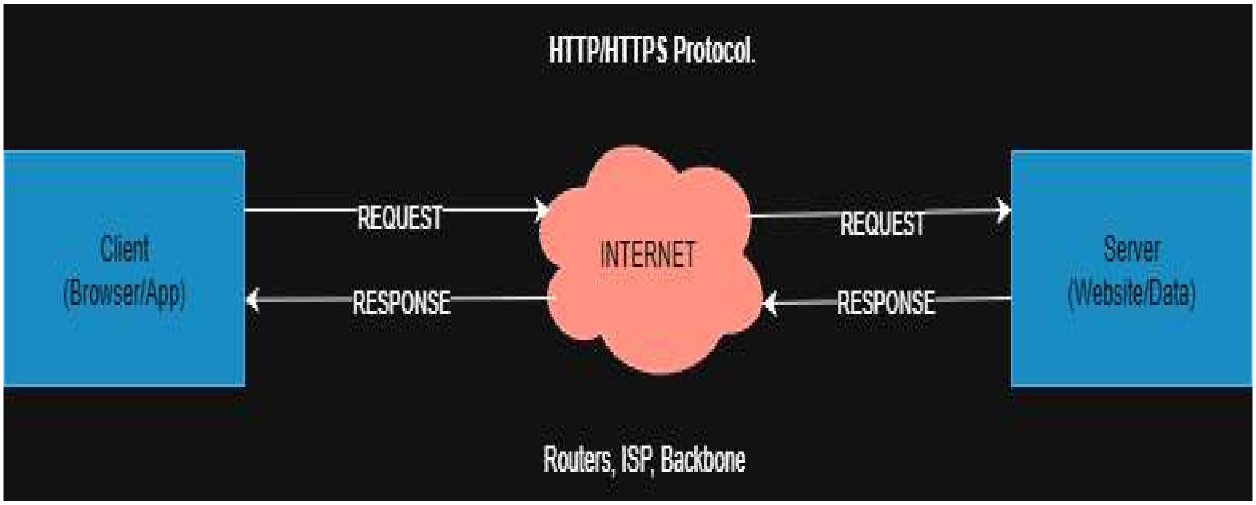
1. Design a simple HTTP client-server communication in any language.

Ans:

Client-Server Communication: How Data is Transmitted Over the Internet.

1. **Client Initiates a Request:**
   - The **client** (e.g., a web browser or mobile app) sends a request to access data or services.
   - This request is usually made using **HTTP or HTTPS** protocols.
2. **Data Travels Through the Internet:**
   - The request is sent through various **routers**, **Internet Service Providers (ISP)**, and the **internet backbone**.
   - Each hop routes the data closer to the **server**.

3. **Server Processes the Request:**
    - The **server** (hosting the website or data) receives the request.
    - The server processes it, accesses its database or application logic, and prepares the **response**.
4. **Response Sent Back to the Client:**
    - The server sends the requested data (HTML, JSON, images, etc.) back to the client.
    - The response follows the reverse path through routers, ISP, and the internet backbone.
5. **Client Displays Data:**
    - The client processes the response and displays the content to the user.



LAB EXERCISE 7:

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

Ans:

1. DSL (Digital Subscriber Line)

- **Description:**

  Uses existing telephone lines to deliver internet.

- **Pros:**
  - Widely available in urban and rural areas.
  - Can use phone line and internet at the same time.
- **Cons:**
  - Speeds slower than cable or fiber.
  - Performance drops with distance from provider's station.

2. Cable Internet

- **Description:**

  Uses coaxial TV cables for internet.

- **Pros:**
  - Faster than DSL (often 100 Mbps+).
  - Widely available in cities and suburbs.
- **Cons:**
  - Speed may drop during peak hours (shared bandwidth).
  - Can be more expensive than DSL.

3. Fiber-Optic Internet

- **Description:**

  Uses thin glass/plastic fibers to transmit data as light.

- **Pros:**
  - Equal upload and download speeds.
  - Very low latency.
  - Reliable and not affected much by weather.

- **Cons:**
    - Limited availability (mostly in cities).
    - Can be more expensive to install.

4. Satellite Internet

- **Description:**

  Uses satellites to provide internet, especially in remote areas.

- **Pros:**
    - Available almost anywhere.
    - Good option for rural or isolated regions.
- **Cons:**
    - High latency (delay in response).
    - Affected by weather.
    - Higher data costs.

5. Mobile Internet (3G/4G/5G)

- **Description:**

  Internet over cellular networks.

- **Pros:**
    - Portable and can be used anywhere with signal.
    - 5G offers very high speeds in covered areas.
- **Cons:**
    - Data limits and higher costs.
    - Signal strength can vary.

6. Fixed Wireless

- **Description:**

  Uses radio signals from a local tower to provide internet to a fixed location.

- **Pros:**
    - Good for areas without cable or DSL.
    - No need for phone or cable lines.
- **Cons:**
    - Requires clear line-of-sight to the tower.
    - Speeds affected by weather and obstructions.

LAB EXERCISE 8:

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

Ans:

HTTP with curl:

```
# GET
curl https://example.com

# GET with headers
curl -i https://example.com

# POST form
curl -X POST -d "name=Hensi&age=21" https://example.com

# POST JSON
curl -X POST -H "Content-Type: application/json" \
-d '{"name":"Hensi"}' https://example.com/api
```

FTP with curl:

```
# Download
curl ftp://ftp.example.com/file.txt --user user:pass -O

# Upload
curl -T local.txt ftp://ftp.example.com/ --user user:pass

# List files
curl ftp://ftp.example.com/ --user user:pass
```

LAB EXERCISE 9:

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

Ans:

Common Application Security Vulnerabilities & Fixes:

1. SQL Injection (SQLi):

- **Meaning**:

  Attacker adds harmful SQL commands into input fields to read, change, or delete database data.

- **Example**:

  ' OR '1'='1

- **Solution**:

  - Use **prepared statements / parameterized queries**.

  - Validate & sanitize inputs.

2. Cross-Site Scripting (XSS):

- **Meaning**:

  Attacker injects JavaScript into a webpage that runs on other users' browsers.

- **Impact**:

  Can steal cookies, redirect users, or show fake content.

- **Solution**:

  - Escape all dynamic content in HTML output.

  - Validate & sanitize user input.

3. Cross-Site Request Forgery (CSRF):

- **Meaning**:

  Attacker tricks a logged-in user into doing an action without their consent.

- **Example**:

  Clicking a hidden link that changes your account password.

- **Solution**:
  - Use **CSRF tokens** in forms.
  - Require re-authentication for sensitive actions.

LAB EXERCISE 10:

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

Ans:

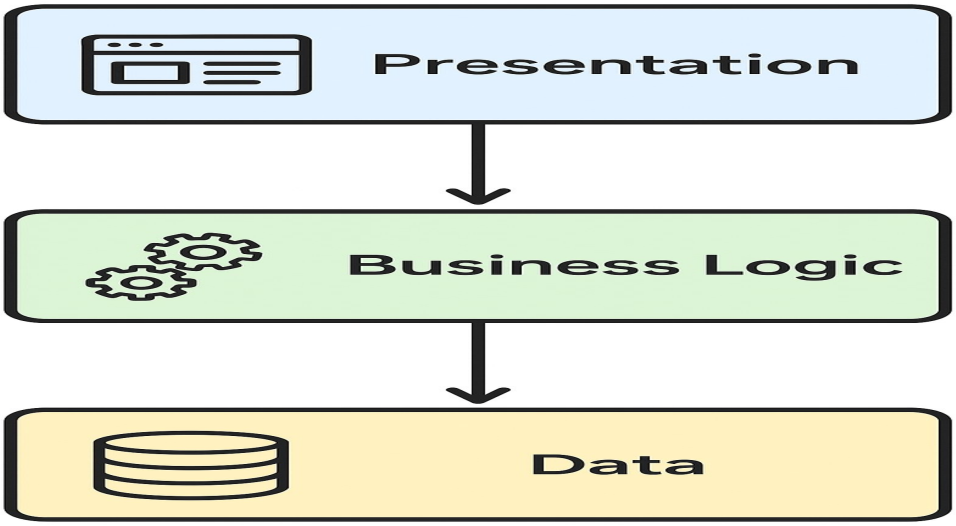
Daily Used Applications

Application Name	Type of Software	Reason
Google Chrome	Application Software	Used for browsing the internet.
Microsoft Word	Application Software	Used for creating and editing documents.
WhatsApp	Application Software	Used for messaging and calling.
Windows 10	System Software	Operating system that manages the computer.
Antivirus	System Software	Protects the system from viruses/malware.

LAB EXERCISE 11:

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

Ans:



Here’s the explanation of the **Three-Tier Architecture** shown in the diagram:

1. Presentation Tier (Client Layer):

- **Role:**

  This is the user interface of the application.

- **Purpose:**

  Displays information to the user and collects input.

- **Examples:**

  Web pages (HTML, CSS, JavaScript), mobile app screens.

- **Key Point:**

  It only handles presentation; no business logic or data storage here.

2. Business Logic Tier (Application Layer):

- **Role:**

  Processes user requests, applies rules, and coordinates data flow between the presentation and data layers.

- **Purpose:**

  Implements the “brains” of the application—validation, calculations, and decisions.

- **Examples:**

  Backend code in Python, Java, PHP, or Node.js.

- **Key Point:**

  Keeps business rules separate from the UI and database, making changes easier.

3. Data Tier (Database Layer):

- **Role:**

  Stores, retrieves, and manages data.

- **Purpose:**

  Ensures data integrity and security.

- **Examples:**

  MySQL, PostgreSQL, MongoDB.

- **Key Point:**

  The application layer interacts with this tier through queries and APIs.

LAB EXERCISE 12:

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

Ans:

Case Study: <u>Online Food Ordering System</u>

- **Presentation Layer:**

  User interface for browsing menu, adding items to cart, entering address/payment, and tracking orders.

  Tech : HTML, CSS, JS, React, Angular.

- **Business Logic Layer:**

  Validates login, applies pricing/tax rules, checks stock, manages payments, updates order status.

  Tech: Python, Java, PHP, Node.js.

- **Data Access Layer:**

  Stores menu, customer data, orders; fetches and updates stock; retrieves past orders.

  Tech: MySQL, PostgreSQL, MongoDB.

<u>Flow:</u>

Customer orders → Business logic processes → Database updates → UI shows confirmation.

How They Work Together:

1. **Customer Action:**

   User selects "Margherita Pizza" and clicks “Order Now” on the website (Presentation Layer).

2. **Business Processing:**

   Application layer validates the request, calculates total price, and checks stock (Business Logic Layer).

3. **Data Handling:**

   The system queries the database for pizza availability and updates stock after order confirmation (Data Layer).

4. **Response Back:**

   Confirmation and estimated delivery time are sent back to the user’s screen (Presentation Layer).

LAB EXERCISE 13:

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

Ans:

Types of Software Environments:

1. Development Environment:

- **Purpose:** Where developers write and test new code.
- **Features:** Debugging tools, local database copies, relaxed security.
- **Example Tools:** VS Code, local servers, Git.

2. Testing Environment:

- **Purpose:** Used by QA teams to verify functionality before release.
- **Features:** Controlled data set, test scripts, bug tracking.
- **Example Tools:** Selenium, JUnit, Postman.

3. Production Environment:

- **Purpose:** Live system used by real customers.
- **Features:** High security, real data, full performance monitoring.
- **Example Tools:** Apache/Nginx, AWS, Azure.

Setting Up a Basic Environment in a Virtual Machine:

We'll use:

- **VirtualBox** (free VM software)
- **Ubuntu Linux** (common for development/testing)

Steps:

1. **Download & Install VirtualBox** → https://www.virtualbox.org
2. **Download Ubuntu ISO** → https://ubuntu.com/download

3. **Create New VM in VirtualBox:**

   - Name: DevTestVM
   - Type: Linux → Ubuntu (64-bit)
   - RAM: 2GB or more
   - Disk: 20GB (dynamically allocated)

4. **Attach Ubuntu ISO** to VM and start.

5. **Install Ubuntu** by following the setup wizard.

6. **Install Development Tools** inside VM:

   ```
   sudo apt update
   sudo apt install git python3-pip mysql-server
   ```

7. **Configure Testing Tools** (optional):

   ```
   pip install pytest selenium
   ```

8. **Simulate Environment Separation**:

   - /home/user/dev → Development files
   - /home/user/test → Testing files
   - Production is simulated as a separate server or another VM.

LAB EXERCISE 14:

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

Ans:

- **Steps to Write and upload your first source code file to Github:**

Step 1: Write your first C source code file:

- Create a file named **hello.c**:

```
#include <stdio.h>
#include<conio.h>

void main()
{
  printf("Hello world!");
}
```

Save this file anywhere (e.g., Desktop or a folder named my-first-c).

Step 2:Create a new GitHub repository:

1. Go to https://github.com and **sign in**.

2. Click **New** (top left).

3. Fill:
   - **Repository name** → first-c-upload
   - **Description** → My first C program upload
   - **Public** selected
   - Leave “Initialize with README” **unchecked**

4. Click **Create repository**.

Step 3:Upload using Gitbash (command line):

If you don’t have Gitbash, download & install it.

Then in **Command Prompt** (Windows):

```
# Go to folder where hello.c is saved
cd path/to/your/folder
```

# Initialize Git in the folder

```
git init
```

# Add your C file to Git tracking

```
git add hello.c
```

# Save the file to Git history with a commit message

```
git commit -m "My first C program upload to GitHub"
```

# Link local folder to GitHub repository (replace YOUR-USERNAME)

```
git remote add origin https://github.com/YOUR-USERNAME/first-c-upload.git
```

# Push the file to GitHub

```
git branch -M main
git push -u origin main
```

step 4:Verify:

Go to your GitHub repository page → refresh → you should see **hello.c** there.

LAB EXERCISE 15:

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

Ans:

1. Create GitHub repository:

- Go to github.com → **New repository.**
- Name it → choose **Public** → click **Create repository.**

2. Local setup:

# Go to your project folder:

```
cd path/to/project
```

# Initialize Git:

```
git init
```

# Add all files:

```
git add .
```

# Commit files:

```
git commit -m "First commit"
```

# Link local folder to GitHub (replace USERNAME and REPO):

```
git remote add origin https://github.com/USERNAME/REPO.git
```

# Push to GitHub:

```
git branch -M main
git push -u origin main
```

LAB EXERCISE 16:

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

Ans:

Link : https://github.com/Hensi200524/student_project

LAB EXERCISE 17:

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

Ans:

1. System Software:

(Helps run the computer hardware and provides a platform for applications)

- **Windows 11** – Operating System
- **Linux Ubuntu** – Operating System
- **macOS** – Operating System
- **Device Drivers** – For printers, graphics cards, etc.

2. Application Software:

(Programs you use to perform specific tasks)

- **Microsoft Word** – Word processing
- **Google Chrome** – Web browsing
- **WhatsApp** – Messaging
- **Zoom** – Video conferencing
- **Spotify** – Music streaming
- **VS Code** – Code editing
- **MS Excel** – Spreadsheets

3. Utility Software:

(Helps maintain, optimize, or protect the system)

- **WinRAR / 7-Zip** – File compression
- **Antivirus Software** – Protection
- **Disk Cleanup Tool** – Remove junk files
- **CCleaner** – System optimization
- **Backup Software** – Data backup

LAB EXERCISE 18:

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

Ans:

```
Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git init
Reinitialized existing Git repository in
C:/Users/Vaghela/Documents/GitHub/student_project/.git/

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git add .

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git commit -m "Initial commit"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git remote add origin https://github.com/Hensi200524/student_project.git
error: remote origin already exists.

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git remote set-url origin https://github.com/Hensi200524/student_project.git

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git push -u origin main
branch 'main' set up to track 'origin/main'.
Everything up-to-date

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git add .

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git commit -m "Your message"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

```
Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git push
Everything up-to-date

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git pull origin main
From https://github.com/Hensi200524/student_project
 * branch            main       -> FETCH_HEAD
Already up to date.

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git add .

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git commit -m "Describe what you changed"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git push origin main
Everything up-to-date

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git add .
```

*merging:

```
Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git commit -m "Resloved conflicts"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
g
Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git push origin main
Everything up-to-date

Vaghela@DESKTOP-Q33QTVR MINGW64 ~/Documents/GitHub/student_project (main)
$ git push origin main --force
Everything up-to-date
```

LAB EXERCISE 19:

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

Ans:

Report on Types of Application Software and Their Role in Improving Productivity:

1. Introduction:

- Application software refers to programs designed to help users perform specific tasks, such as creating documents, managing data, or communicating with others.
- Unlike system software, which runs the computer, application software directly assists users in achieving work-related, creative, or personal goals.

- The right application software significantly improves productivity by automating tasks, enhancing accuracy, and enabling collaboration.

2. Types of Application Software:

a) Word Processing Software:

- **Examples:** Microsoft Word, Google Docs.
- **Purpose:** Create, edit, and format text-based documents.
- **Productivity Benefit:** Speeds up document creation, allows quick formatting, enables collaboration through cloud sharing.

b) Spreadsheet Software:

- **Examples:** Microsoft Excel, Google Sheets
- **Purpose:** Organize, calculate, and analyze numerical data.
- **Productivity Benefit:** Automates calculations, generates charts, supports data analysis for decision-making.

c) Database Management Software (DBMS):

- **Examples:** MySQL, Microsoft Access, Oracle Database
- **Purpose:** Store, organize, and manage large volumes of data.
- **Productivity Benefit:** Improves data retrieval speed, ensures data accuracy, and supports complex queries.

d) Presentation Software:

- **Examples:** Microsoft PowerPoint, Google Slides
- **Purpose:** Create slideshows for meetings, teaching, or training.
- **Productivity Benefit:** Makes information visually engaging, improves communication, and saves time in presenting ideas.

e) Web Browsers:

- **Examples:** Google Chrome, Mozilla Firefox, Microsoft Edge
- **Purpose:** Access and navigate the internet.
- **Productivity Benefit:** Provides instant access to information, supports web-based applications, and enables online collaboration.

f) Communication Software:

- **Examples:** Zoom, Microsoft Teams, Slack
- **Purpose:** Facilitate real-time communication and file sharing.
- **Productivity Benefit:** Reduces travel time, enables remote work, and speeds up team coordination.

g) Graphic Design & Multimedia Software:

- **Examples:** Adobe Photoshop, Canva, CorelDRAW
- **Purpose:** Create and edit images, videos, and animations.
- **Productivity Benefit:** Speeds up creative projects, improves quality of visuals, and allows faster content production.

h) Project Management Software:

- **Examples:** Trello, Asana, Microsoft Project
- **Purpose:** Plan, track, and manage projects.
- **Productivity Benefit:** Keeps tasks organized, monitors progress, and ensures deadlines are met.

i) Specialized Industry Software:

- **Examples:** AutoCAD (engineering), Tally ERP (accounting), MATLAB (science)
- **Purpose:** Address industry-specific tasks.
- **Productivity Benefit:** Increases efficiency in specialized workflows.

3. How Application Software Improves Productivity:

1. **Automation of Repetitive Tasks**:

   Reduces manual effort, e.g., formulas in Excel.

2. **Enhanced Collaboration**:

   Cloud-based tools allow real-time teamwork.

3. **Time Savings**:

   Faster document creation, data processing, and communication.

4. **Accuracy & Consistency**:

   Built-in tools reduce human error.

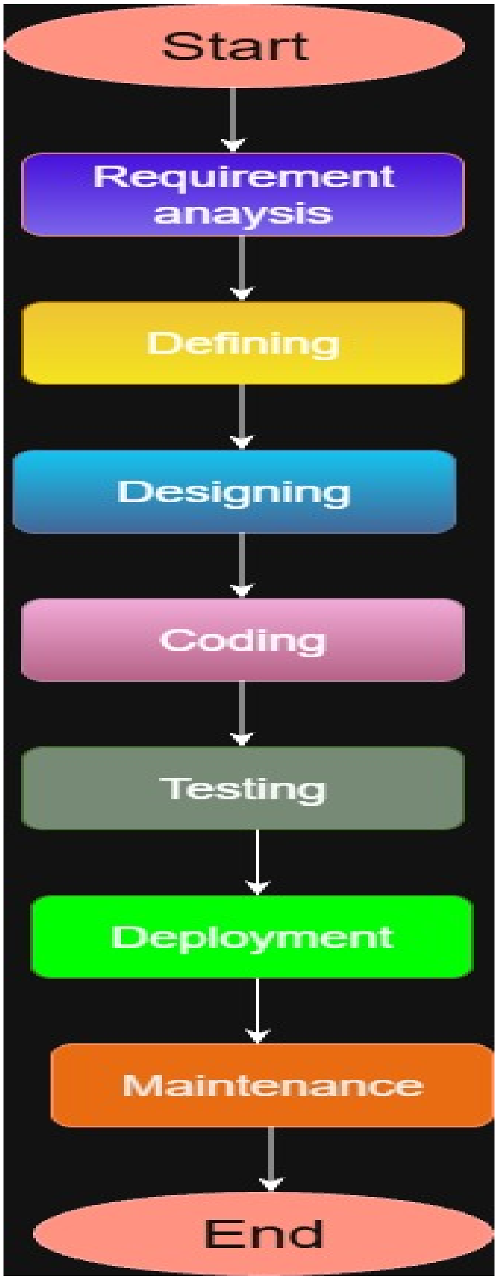
5. **Better Decision-Making**:

   Data visualization and analysis improve insights.

LAB EXERCISE 20:

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

Ans:



LAB EXERCISE 21:

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

Ans:

Requirement Specification – Library Management System:

1. Functional Requirements:

- **User Management:** Register, update, and delete member details.
- **Book Management:** Add, update, and remove book records.
- **Borrow/Return:** Issue books, track due dates, and update availability.
- **Search:** Find books by title, author, or category.
- **Fine Management:** Calculate and record late return fines.

2. Non-Functional Requirements:

- Easy-to-use interface.
- Fast processing.
- Secure admin login.
- Daily data backup.

3. Constraints:

- Platform: Web or desktop.
- Database: MySQL.

LAB EXERCISE 22:

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

Ans:

Functional Analysis – Online Shopping System

1. User Functions:

- **User Registration & Login**:

  Create an account, log in securely.

- **Browse Products**:

  View products by category, price, or search keywords.

- **Product Details**:

  View descriptions, images, prices, and reviews.

- **Shopping Cart**:

  Add, remove, or update product quantities.

- **Checkout & Payment**:

  Place orders with multiple payment options (credit/debit card, UPI, COD).

- **Order Tracking**:

  View order status and delivery updates.

- **Reviews & Ratings**:

  Post feedback for purchased products.

2. Admin Functions:

- **Product Management**:

  Add, edit, or delete products.

- **Inventory Management**:

  Track stock levels.

- **Order Management**:

  Process, ship, and update order statuses.

- **User Management**:

  Manage customer accounts.

- **Reports**:

  Sales, revenue, and user activity reports.

3. Non-Functional Aspects:

- **Performance:**

  Fast product searches and page loading.

- **Security:**

  Secure payment processing and encrypted user data.

- **Scalability:**

  Support multiple users simultaneously.

- **Availability:**

  24/7 uptime.

LAB EXERCISE 23:

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

Ans:

1. Main Components:

A) Frontend (Presentation Layer):

- **Customer App**:

  Browse restaurants, place orders, track delivery.

- **Restaurant Dashboard**:

  Manage menu, accept orders.

- **Delivery Partner App**:

  Accept and deliver orders.

B) Backend (Business Logic Layer):

- **Order Management System**:

  Handles order creation, updates, and tracking.

- **Menu & Restaurant Management**:

  Stores restaurant data, menus, and prices.

- **Delivery Management System**:

  Assigns orders to delivery partners.

- **Payment Gateway Integration**:

  Processes online payments securely.

- **Notification Service**:

Sends order updates via push notifications/SMS.

C) Database (Data Layer):

- **User Database**:

  Customer, restaurant, and delivery partner profiles.

- **Menu Database**:

  Items, prices, availability.

- **Order Database**:

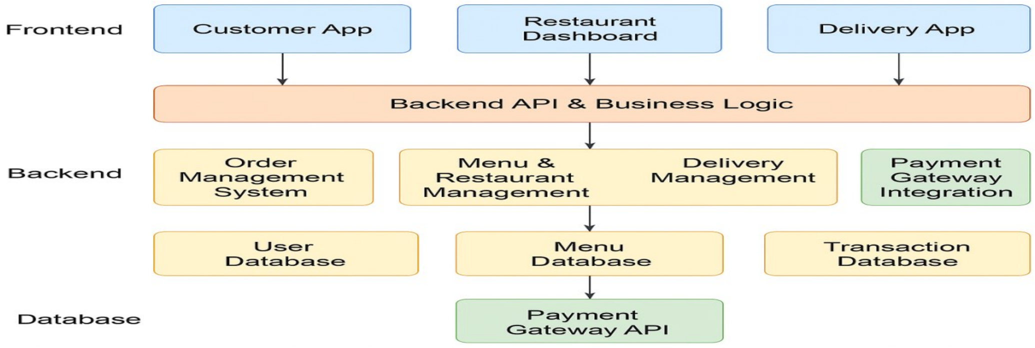
  Order details, statuses, timestamps.

- **Transaction Database**:

  Payment records, receipts.

2. Flow Overview:

1. **Customer places order** → Frontend sends data to backend API.
2. **Backend processes order** → Stores in Order DB and sends confirmation.
3. **Restaurant receives order** → Prepares food.
4. **Delivery partner assigned** → GPS tracking activated.
5. **Payment processed** → Online or Cash on Delivery.
6. **Order delivered** → Status updated in system.



LAB EXERCISE 24:

1.Develop test cases for a simple calculator program.

Ans:

Test Case ID	Test Scenario	Input	Expected Output	Remarks
TC01	Addition of two positive numbers	5 + 3	8	Pass if correct sum
TC02	Addition of positive and negative number	7 + (-4)	3	Check sign handling
TC03	Subtraction of two numbers	10 - 6	4	Basic subtraction
TC04	Subtraction resulting in negative	4 - 9	-5	Negative result
TC05	Multiplication of two numbers	6 × 5	30	Basic multiplication
TC06	Multiplication by zero	9 × 0	0	Zero handling
TC07	Division of two numbers	8 ÷ 2	4	Basic division
TC08	Division by zero	7 ÷ 0	Error / Infinity	Must handle error
TC09	Decimal addition	2.5 + 3.1	5.6	Floating-point precision
TC10	Large number multiplication	100000 × 1000	100000000	Large value handling

LAB EXERCISE 25:

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

Ans:

Real-World Case:

WhatsApp Outage in 2022

What Happened:

In October 2022, WhatsApp stopped working for around two hours worldwide. People couldn’t send or receive messages.

Reason for Maintenance:

A technical problem in WhatsApp’s servers caused the service to fail. Engineers had to fix the issue quickly so people could use the app again.

Type of Maintenance:

- **Corrective Maintenance**:

  fixing a problem after it happened.

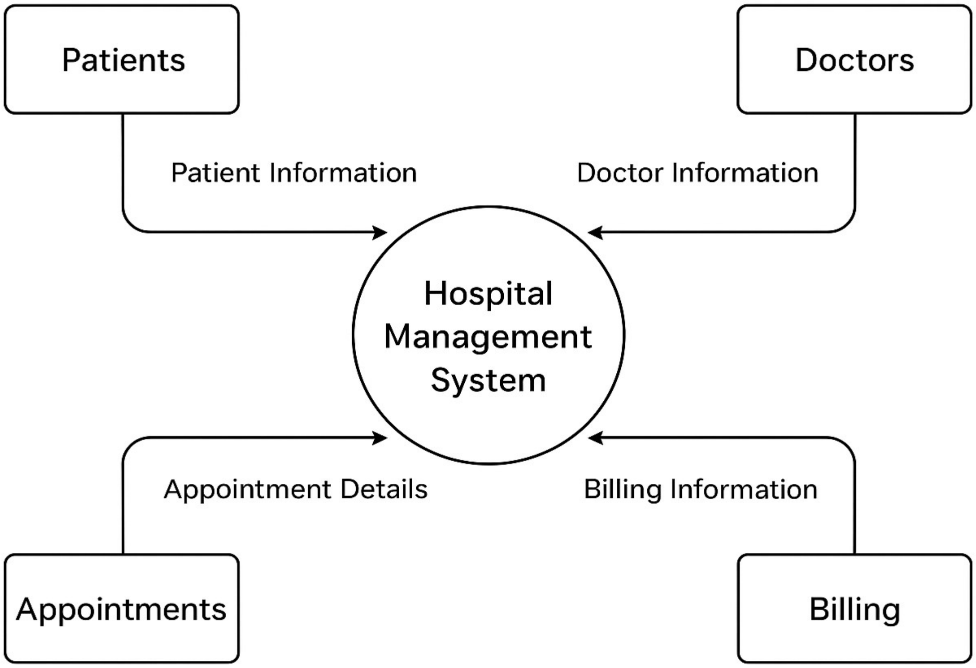
Result:

After maintenance, WhatsApp started working again, and millions of users could send messages as usual.

LAB EXERCISE 30:

1.Create a DFD for a hospital management system.

Ans:



this DFD shows how a Hospital Management System works:

- **Patients** give their details to the system.
- **Doctors** provide doctor information.
- **Appointments** send appointment details.
- **Billing** sends billing details.

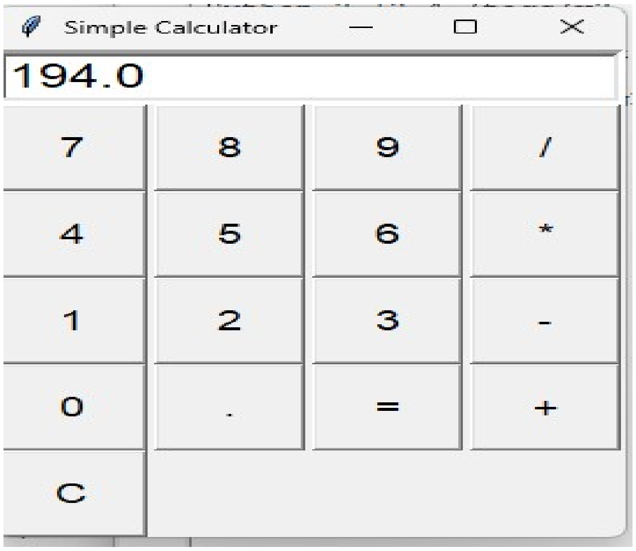
LAB EXERCISE 31:

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

Ans:

```
import tkinter as tk

def click(button_text):

    if button_text == "=":

        try:

            result = eval(entry.get())

            entry.delete(0, tk.END)

            entry.insert(tk.END, str(result))

        except:

            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)

root = tk.Tk()

root.title("Simple Calculator")

entry = tk.Entry(root, width=20, font=("Arial", 18), borderwidth=5)

entry.grid(row=0, column=0, columnspan=4)
```

```
buttons = [
    "7", "8", "9", "/",
    "4", "5", "6", "*",
    "1", "2", "3", "-",
    "0", ".", "=", "+",
    "C" ]
row, col = 1, 0
for btn in buttons:
    tk.Button(root, text=btn, width=5, height=2, font=("Arial", 14),
              command=lambda b=btn: click(b)).grid(row=row, column=col)
    col += 1
    if col > 3:
        col = 0
        row += 1
root.mainloop()
```

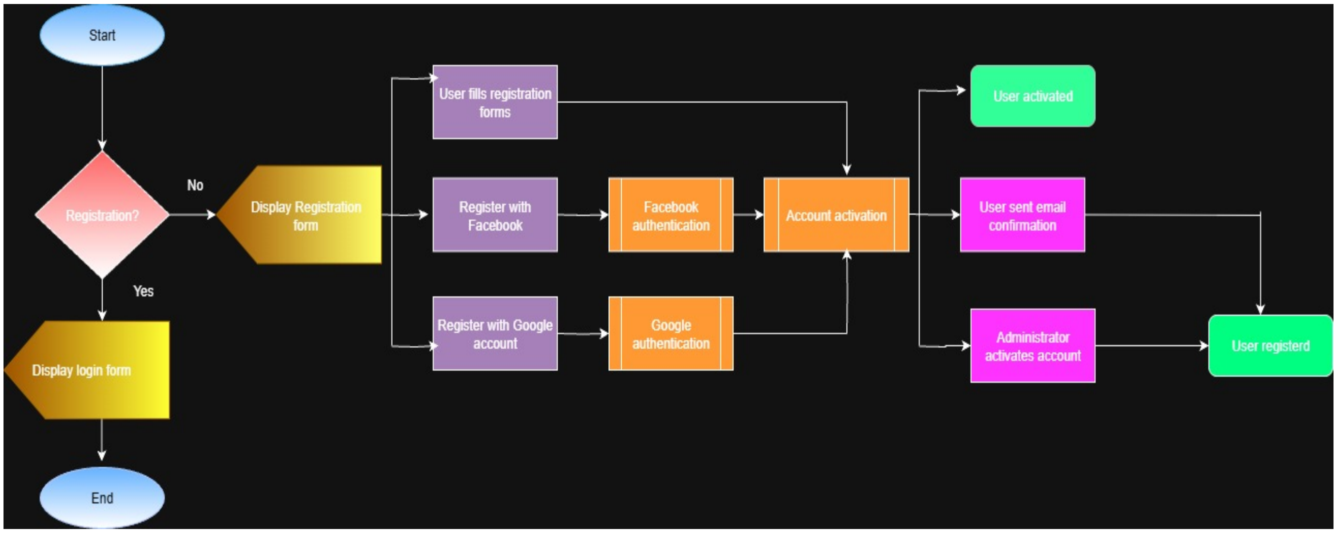


194.0

LAB EXERCISE 32:

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

Ans:



Explain:

This flowchart shows the steps for a **user registration and login process**:

1. **Start** – Process begins.
2. **Decision: Registration?**
   - If **Yes** → Display login form → End.
   - If **No** → Go to registration process.
3. **Display Registration Form** – User chooses one method:
   - **Fill forms manually**.
   - **Register with Facebook** → Facebook authentication.
   - **Register with Google** → Google authentication.
4. **Account Activation** – After authentication, the account is activated.
5. **Activation Methods**:
   - Direct activation → **User activated**.
   - Email confirmation → Admin approval → **User registered**.