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

C language

#include<stdio.h>

main()

{

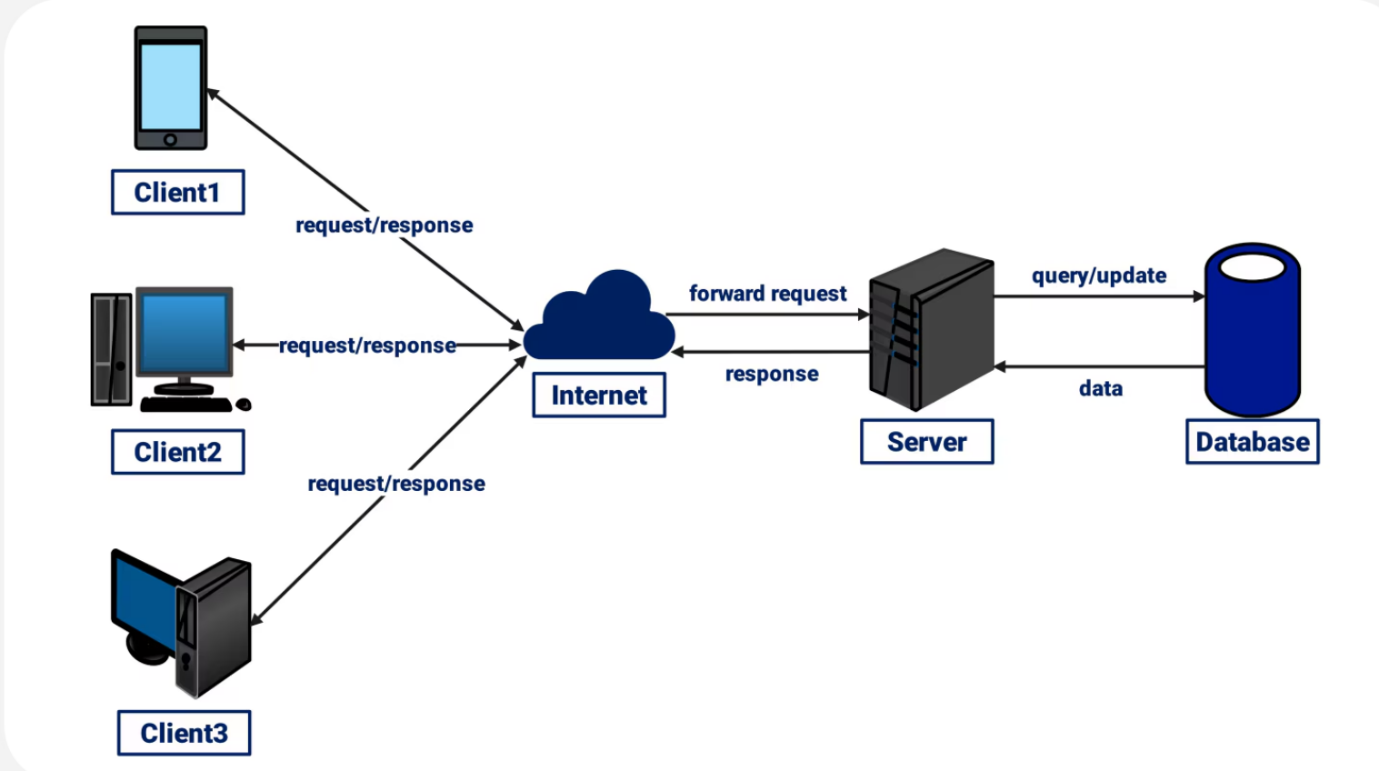
printf("Hello world");

}

Python

Print(“hello world”)

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



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

**Client-server architecture** is a **network model** in which two main entities — **clients** and **servers** — communicate with each other to complete specific tasks or share data.

* **Client**: The client initiates requests, waits for the server’s response, and displays it to the user. For example, a web browser acts as a client when it requests a website.
* **Server**: The server processes these requests, retrieves the relevant information, and sends it back to the client. For example, a web server responds with website data when a browser requests it.

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

**1. broadband**

**Pros**:

* High-speed internet
* Supports multiple devices

**Cons**:

* Can be expensive
* Some plans have data limits

**2.Fiber Optic Internet**

**Pros**:

* Ultra-fast speeds
* Highly reliable and low latency

**Cons**:

* Expensive installation
* Limited availability, especially in rural areas

**3.Satellite Internet**

**Pros**:

* Available in remote and rural locations
* No need for physical cables

**Cons**:

* Weather can affect signal
* Data caps and higher cost

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

To simulate HTTP and FTP requests using command line tools, we use commands like curl. HTTP is used to request web pages or data from a server, while FTP is used to upload or download files. These tools help us understand how data is transferred between a client and a server over the internet.

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

**🔒 Cryptographic Failures**

**What:** Sensitive data exposed due to weak encryption, poor key management, or no encryption.  
**Fix:** Use strong, standard cryptography (TLS, AES‑GCM), secure key storage, and proper encryption for data at rest and in transit .

**🧩  Insecure Design**

**What:** Security issues originate from design flaws—not implementation bugs—like flawed logic, missing threat modeling, or lack of secure architecture.  
**Fix:** Apply threat modeling, adopt secure design patterns, include security in early design/user stories .

**⚙️ Vulnerable & Outdated Components**

**What:** Use of unsupported, outdated, or vulnerable libraries/frameworks (including nested dependencies) leads to known exploits like Log4Shell.  
**Fix:** Maintain an inventory of components, regularly scan and update dependencies, remove unused code, subscribe to CVE alerts .

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

**Google Chrome – Application Software (used for browsing the internet)**

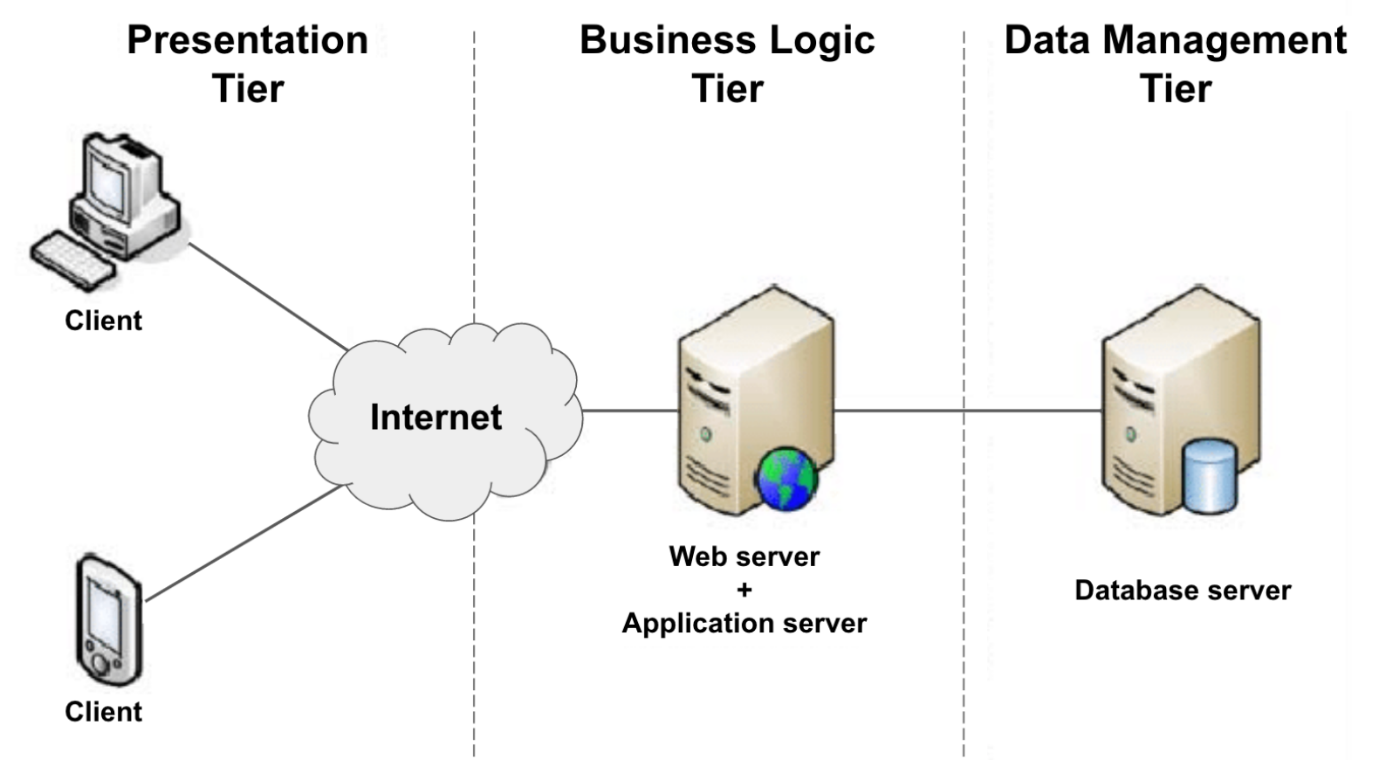
**Microsoft Word – Application Software (used for creating documents)**

**Windows OS – System Software (controls and manages the computer system)**

**File Explorer – System Software (used to manage files and folders)**

**WhatsApp – Application Software (used for messaging and communication)**

LAB EXERCISE: Design a basic three-tier software architecture diagram for a web application.



So, the architecture of modern web applications adheres to the foundational 3-tier concept. This architectural paradigm systematically divides applications into three distinct tiers:

* The presentation tier (client layer) that is responsible for user interface interactions ;
* The application tier (business layer), managing core application functionality ;
* The data tier handles the storage and retrieval of information.

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

**Case Study Example: Online Library System**

**1. Presentation Layer (User Side)**

* This is what the user sees: web pages to search books, login, or borrow books.
* Example: A student opens the library website and searches for a book.

**2. Business Logic Layer (Processing Side)**

* This checks rules: Is the book available? Is the user logged in?
* Example: System checks if the book is already borrowed or not.

**3. Data Access Layer (Database Side)**

* This layer talks to the database: gets book data, updates borrow status.
* Example: System marks the book as "borrowed" in the database.

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

**Development**

* Where developers write code, run unit tests, and debug locally.
* Tools: IDEs, local databases, dev servers.

**Testing (QA / Integration)**

* Used to test features, modules, integration, performance, and security. Simulates production closely.

**Staging (Pre‑Production)**

* A full replica of production for final validation and user acceptance testing before deployment.

**Production**

* The live environment serving real users; needs high stability, security, monitoring, and minimal downtime.

LAB EXERCISE: Create a Github repository and document how to commit and push code changes.

**Step 1: Create a GitHub Repository**

1. **Go to GitHub:** <https://github.com>
2. **Log In** to your GitHub account.
3. **Create a new repository:**
   * Click the **+** icon in the top-right corner.
   * Choose **"New repository"**.
4. **Fill in repository details:**
   * **Repository name:** example-repo (use your preferred name)
   * **Description:** (optional) *A sample repository to demonstrate Git workflow.*
   * **Visibility:** Public or Private
   * (Optional) Check **Initialize this repository with a README**
5. Click **"Create repository"**

**Step 2:Commit the Changes**

1.Now commit the staged changes with a meaningful message:

If nothing is staged, this command won’t do anything.

**Step 3: Push the Changes to GitHub**

Now that the changes are committed locally, you push them to GitHub

git push origin main

If you're on a different branch, like dev, use:

git push origin dev

If it's the first push for a new branch:

git push -u origin dev

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

**System Software**

* Windows 10 / 11 (Operating System)
* Device Drivers (e.g., printer or graphics driver)

**Application Software**

* Microsoft Word (Word Processing)
* Google Chrome (Web Browser)

**Utility Software**

* Disk Cleanup / Disk Defragmenter
* Backup Tools (e.g., Acronis, Windows Backup)

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

|  |  |
| --- | --- |
| Task | Command Example |
| Clone | git clone URL |
| branch | git checkout -b feature-name |
| Merge | git checkout main → git merge branch |

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

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

1. **Word Processing Software**
   * **Example**: Microsoft Word, Google Docs
   * **Use**: Writing documents, letters, reports
   * **Productivity**: Speeds up writing with tools like auto-correct, templates, and collaboration.
2. **Spreadsheet Software**
   * **Example**: Microsoft Excel, Google Sheets
   * **Use**: Data analysis, financial planning, reports
   * **Productivity**: Automates calculations and data organization; useful in business and accounting.
3. **Presentation Software**
   * **Example**: Microsoft PowerPoint, Prezi
   * **Use**: Creating visual slides for meetings or lectures
   * **Productivity**: Enhances communication of ideas clearly and quickly.
4. **Database Management Software**
   * **Example**: Oracle, MS Access, MySQL
   * **Use**: Storing and managing large datasets
   * **Productivity**: Ensures fast data access and organization for decision-making.
5. **Multimedia Software**
   * **Example**: VLC Media Player, Adobe Premiere Pro
   * **Use**: Creating and editing audio, video, and images
   * **Productivity**: Supports creative tasks and professional media production.
6. **Communication Software**
   * **Example**: Zoom, Microsoft Teams, Slack
   * **Use**: Messaging, video conferencing, file sharing
   * **Productivity**: Improves team collaboration and reduces time spent on meetings.
7. **Web Browsers**
   * **Example**: Chrome, Firefox
   * **Use**: Internet access for research and online tools

# LAB EXERCISE: Write a requirement specification for a simple library management system.

**Library Management System – Requirement Specification**

**1. Purpose:**  
 To automate book management, issue, return, and member handling in a library.

**2. Users:**

**Admin/Librarian**: Manage books and members

**Students/Members**: Search and view books

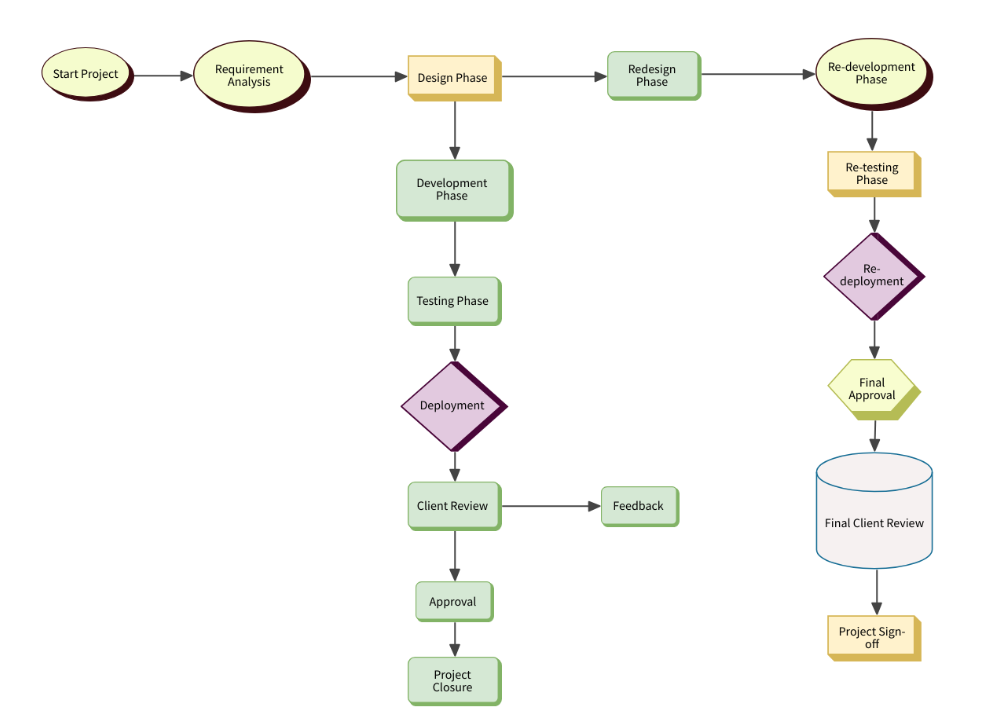
**3. Main Features:**

* Login system for admin
* Add/edit/delete books
* Register/edit/remove members
* Issue and return books
* Search books by title/author
* Generate reports

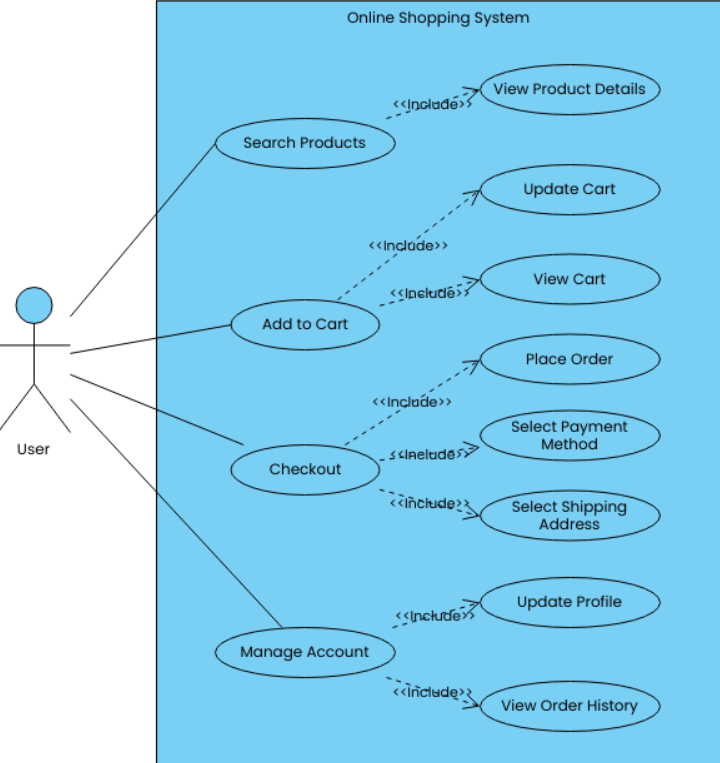
**4. System Requirements:**

* Simple user interface
* Fast access and search
* Role-based access for security

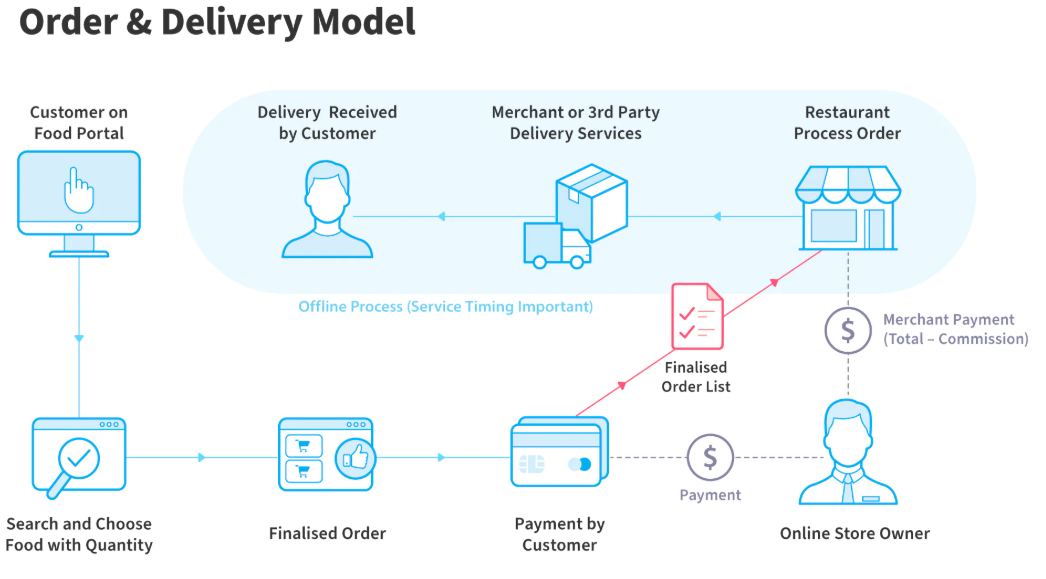
LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDLC).



LAB EXERCISE: Perform a functional analysis for an online shopping system.



LAB EXERCISE: Design a basic system architecture for a food delivery app.



LAB EXERCISE: Develop test cases for a simple calculator program.

**Functional Test Cases For Simple Calculator**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Testcase\_ID** | **Test Cases** |
| 1 | Fun\_Basic\_01 | Verify that the result of the addition operation of two integer numbers is displayed as expected or not |
| 2 | Fun\_Basic\_02 | Verify that the result of the subtraction operation of two integer numbers is displayed as expected or not |
| 3 | Fun\_Basic\_03 | Verify that the result of the multiplication operation of two integer numbers is displayed as expected or not |
| 4 | Fun\_Basic\_04 | Verify that the result of the division operation of two integer numbers is displayed as expected or not |
| 5 | Fun\_Basic\_05 | Verify that the user is able to clear the screen or not |
| 6 | Fun\_Basic\_06 | Verify that the user is able to clear a single digit by backspace or not |
| 7 | Fun\_Basic\_07 | Check that maximum numbers are displayed properly in the LCD screen or not |

LAB EXERCISE: Document a real-world case where a software application required critical maintenance.

What happened?

Facebook, Instagram, and WhatsApp suddenly stopped working worldwide for almost 6 hours.

This happened because Facebook's engineers were doing a routine update to their system, but they accidentally disconnected Facebook from the internet.

Why was it critical?

• No one could access Facebook services.

• Even Facebook's own employees couldn’t log in or fix things remotely.

• Billions of users and businesses were affected.

How did they fix it?

• Engineers had to go to the physical data centers to fix the problem manually.

• They restarted servers and rebuilt the system from scratch.

What did they learn?

• Always test updates more carefully.

• Have backup tools that work even if the main system is down.

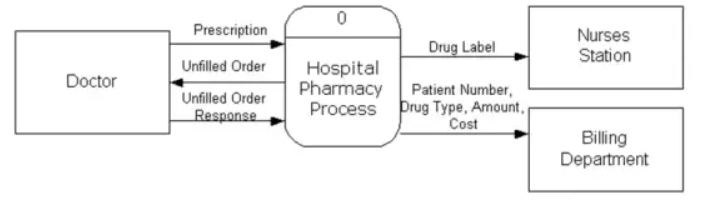
• Make sure updates don’t block access to everything at once.

Impact:

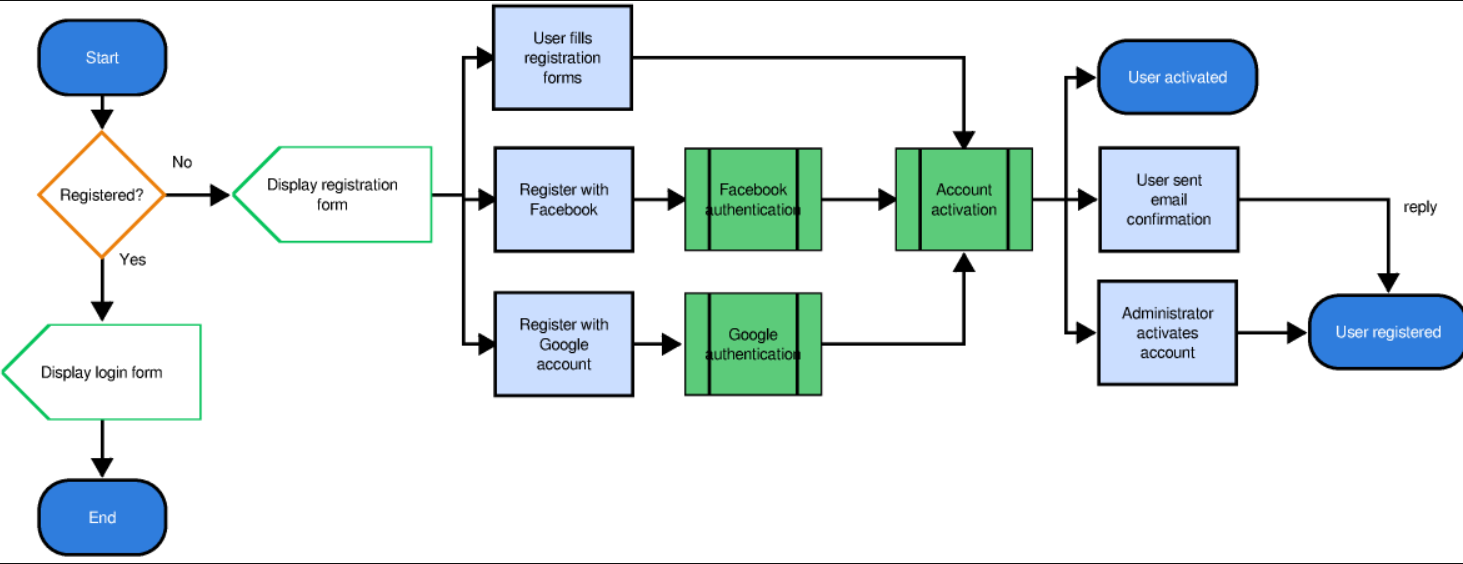
• Huge losses for Facebook (millions of dollars).

• People and businesses who rely on their apps were badly affected.

LAB EXERCISE: Create a DFD for a hospital management system.



LAB EXERCISE: Draw a flowchart representing the logic of a basic online registration system.



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

1. Development Environment

o This is where programmers write and create the software.

o It has tools like code editors and debuggers.

o It’s flexible and changes often.

2. Testing Environment

o After writing the software, it is moved here to check if everything works well.

o Bugs are found and fixed in this environment.

o It tries to copy the real-world setup but isn’t live.

3. Production Environment

o This is the real, live environment where the software is used by customers.

o It is stable and reliable.

o Changes here are made carefully to avoid breaking things.

• How to Set Up a Basic Environment in a Virtual Machine(VM)

1. Choose a Virtual Machine software:

Examples: VirtualBox, VMware, or Hyper-V.

2. Download an Operating System (OS):

example, get an ISO file of Ubuntu (Linux) or Windows.

3. Create a New Virtual Machine:

o Open the VM software.

o Create a new VM and assign memory (RAM), disk space, and CPU cores.

o Attach the OS ISO file to the VM.

4. Start the Virtual Machine:

o Boot up the VM with the OS installer.

o Follow the steps to install the OS inside the VM.

5. Set Up Your Environment:

o Install software like code editors (e.g., VS Code), programming languages, or tools needed.

o This can be your development environment inside the VM.

6. Use Snapshots:

o Save the VM state so you can return to it if needed (useful for testing).

**LAB EXERCISE**: Build a simple desktop calculator application using a GUI library.

#include <stdio.h>

int main()

{

double num1, num2, result;

char op;

printf("Simple Calculator\n");

printf("Enter expression (e.g., 3 + 4): ");

scanf("%lf %c %lf", &num1, &op, &num2);

switch (op)

{

case '+':

result = num1 + num2;

printf("Result: %.2lf\n", result);

break;

case '-':

result = num1 - num2;

printf("Result: %.2lf\n", result);

break;

case '\*':

result = num1 \* num2;

printf("Result: %.2lf\n", result);

break;

case '/':

if (num2 == 0) {

printf("Error: Division by zero\n");

} else {

result = num1 / num2;

printf("Result: %.2lf\n", result);

}

break;

default:

printf("Error: Invalid operator\n");

}

return 0;

}