PYTHON – BACKEND ASSIGNMENT

MODULE 1 – OVERVIEW OF IT INDUSTRY

1.What is a Program:

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

```c
#include <stdio.h>
main()
 {
    printf("Hello, World!\n");
    return 0;
}
```

- **THEORY EXERCISE**: Explain in your own words what a program is and how it functions.

A program is a set of instructions written in a programming language that tells a computer how to perform a specific task or solve a problem.

=>Here's a basic overview of how a program functions:

1.input

2.processing

3.output

2. What is Programming:

❖ **THEORY EXERCISE**: What are the key steps involved in the programming process?

Programming is the process of writing instructions (code) in a specific language that a computer can understand and execute to perform tasks or solve problems.

=>the key steps involved in the programming process:

1.problem definition and planning

2.design

3.writing the code

4.testing

5.refinement and optimization

6.documentation

7.deployment

8.maintenance and updates

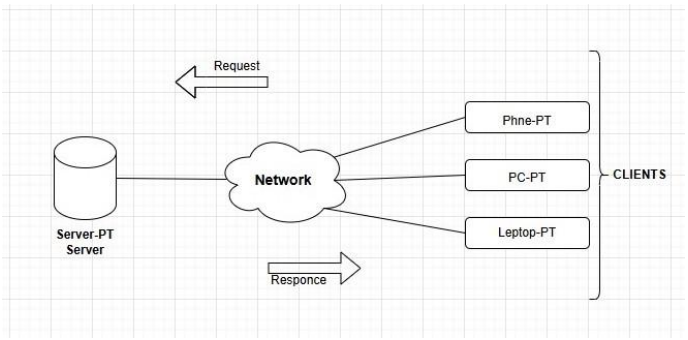
3. Types of Programming Languages :

❖ **THEORY EXERCISE**: What are the main differences between high-level and low-level programming languages?

HIGH-LEVEL PROGRAMMING LANGUAGE	LOW-LEVEL PROGRAMMING LANGUAGE
It is programmer friendly language.	It is a machine friendly language.
It is easy to understand.	It is tough to understand.
It is portable.	It is non-portable.
It can run on any platform.	It is machine-dependent.
Debugging is easy	Debugging is complex comparatively.

4.World Wide Web & How Internet Works :

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

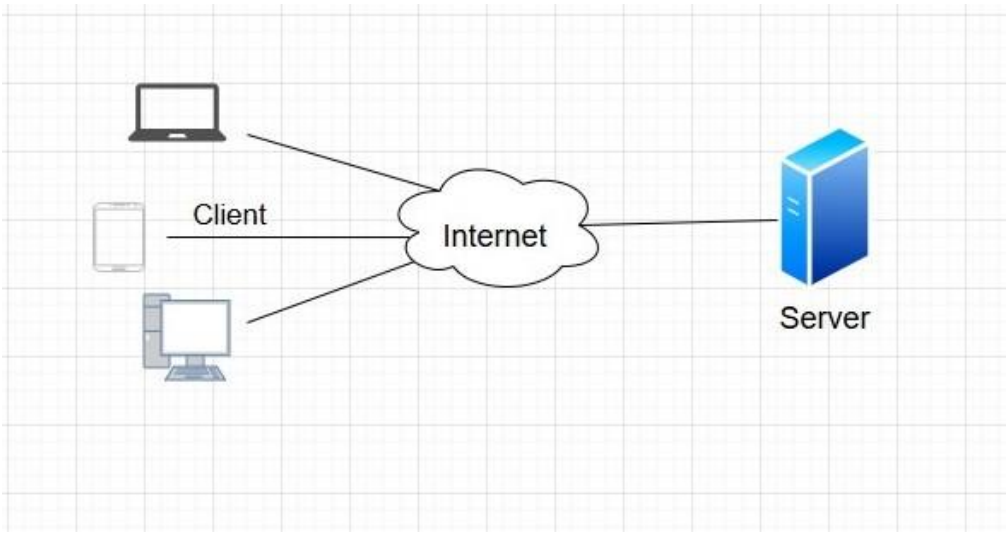


- **THEORY EXERCISE**: Describe the roles of the client and server in web communication

Role	Client Responsibilities	Client Responsibilities
Request	Sends requests for resources (e.g., web pages, data) to the server	Listens for incoming requests and processes them
User Interface	Provides the interface for the user to interact with (e.g., buttons, forms)	May provide dynamic content to update the user interface (via API calls)
Data Rendering	Renders received data to the user (e.g., displaying HTML)	Sends data (e.g., HTML, images, JSON) in response to requests
User Interaction	Collects and sends user inputs (e.g., form submissions)	Processes user inputs, stores data, and may send back a response
Security	Secures user information on the client side (e.g., using HTTPS)	Authenticates users and ensures secure transmission of data

5. Network Layers on Client and Server:

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



❖ **THEORY EXERCISE:** Explain the function of the TCP/IP model and its layers.

- The function of the TCP/IP model and its layers:
  1. Application Layer:
     Provides network services to end-user applications.

  2.Transport Layer:
     Ensures reliable data transfer, error detection, and flow control.

  3.Internet Layer:
     Handles logical addressing and routing of data between networks.

  4.Link Layer:
     Manages physical transmission of data over a local network.

6. Client and Servers:

❖ **THEORY EXERCISE**: Explain Client Server Communication.

- **Synchronous Communication**: The client sends a request and waits for the server's response before proceeding. Common in most web and application interactions.
- **Asynchronous Communication**: The client sends a request and continues with other tasks without waiting for the response. The server responds later, and the client processes the response when it arrives. This is typical in messaging systems or APIs where the client doesn't need an immediate response.

7. Types of Internet Connections:

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

Type of Connection	**Pros**	Cons
Broadband (DSL)	Affordable, widely available, stable	Slow speeds, distance-sensitive
Cable Internet	Faster speeds, more stable than DSL	Shared bandwidth, more expensive than DSL
Fiber-Optic	Ultra-fast speeds, reliable, symmetrical speeds	Expensive, limited availability, installation time
Satellite	Available in remote areas, global reach	High latency, slow speeds, weather-sensitive
Mobile Hotspot/4G	Portable, quick setup	Data limits, slower speeds, coverage limitations

- **THEORY EXERCISE:** How does broadband differ from fiber-optic internet?

Feature	Broadband	Fiber-Optic Internet
Technology	DSL, Cable, Satellite, Wireless	fiber-optic cables (glass or plastic fibers)
Speed	10 Mbps to 1 Gbps, depending on the tech	100 Mbps to 10 Gbps or more
Latency	Higher, especially with satellite	Extremely low latency
Cost	Less expensive, depending on tech	More expensive due to installation costs
Installation	Easier installation (DSL, cable)	More complex installation (fiber cables)

8. Protocols:

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

  - **Simulating HTTP Requests with curl:**

- GET Request (Fetch Data): A simple GET request fetches data from a URL.
- POST Request (Submit Data): A POST request is typically used to submit data to a server (e.g., form data).

- GET Request with Custom Headers: You can also send custom headers with your HTTP request.
- GET Request with Query Parameters: You can pass query parameters in the URL.
- Download a File: You can download a file using curl with the -O option.
- HTTP GET Request with Verbose Output: If you want to see detailed information about the request/response, use the -v option

➢ **Simulating FTP Requests with curl:**

- FTP GET Request (Download a File): You can download a file from an FTP server using curl with FTP credentials.
- FTP PUT Request (Upload a File): You can upload a file to an FTP server with curl by using the -T option.
- FTP with Passive Mode: Sometimes, FTP servers require passive mode for data transfer. You can enable passive mode with the --ftp-pasv option
- FTP Listing (List Files in a Directory): To list files in an FTP directory:

❖ **THEORY EXERCISE**: What are the differences between HTTP and HTTPS protocols?

Aspect	HTTP	HTTPS
Security	No encryption; data sent in plain text	Data is encrypted with SSL/TLS
Port	Uses port 80	Uses port 443
SSL/TLS Certificate	No certificate required	Requires an SSL/TLS certificate
SEO Ranking	Less favored by search engines	Favored by search engines (better ranking)
Common Usage	Informational websites or non-sensitive data	Websites handling sensitive data (banking, shopping)

9. Application Security:

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

- **SQL Injection (SQLi):** SQL Injection occurs when an attacker is able to manipulate SQL queries by injecting malicious SQL code through user inputs. This vulnerability allows attackers to bypass authentication, retrieve sensitive information, modify or delete data, and in some cases, gain administrative access to the database.
- **Cross-Site Scripting (XSS):** Cross-Site Scripting (XSS) occurs when an attacker injects malicious scripts (usually JavaScript) into web pages viewed by other users.
- **Cross-Site Request Forgery (CSRF):** Cross-Site Request Forgery (CSRF) attacks trick the user into unknowingly submitting a request to a web application where the user is authenticated, such as changing their password or transferring funds.

❖ **THEORY EXERCISE:** What is the role of encryption in securing applications?

- **Here are the primary roles of encryption in securing applications:**

1. Data Privacy and Confidentiality:
2. Data Integrity:
3. Authentication and Identity Verification:
4. Secure Communication (Data Transmission):
5. Protection of Stored Data (At-Rest Encryption):
6. Compliance with Regulations:
7. Preventing Unauthorized Access and Data Breaches:
8. Role in Cloud and Multi-Tenant Environments:

10. Software Applications and Its Types:

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

➢ **system software:**

- **Operating System (e.g., Windows, macOS, Linux):** The operating system (OS) is system software that manages hardware and software resources and provides common services for computer programs.
- **Device Drivers (e.g., Printer Driver, Graphics Card Driver):** : Device drivers are system software that allow the operating system to communicate with hardware devices (e.g., printers, graphics cards).

➢ **Application Software:**

- **Microsoft Office (e.g., Word, Excel):** Microsoft Office is a suite of application software that provides various productivity tools like word processing, spreadsheets, and presentations.
- **Web Browser (e.g., Google Chrome, Mozilla Firefox):** A web browser is an application software that allows users to access, retrieve, and view content from the internet

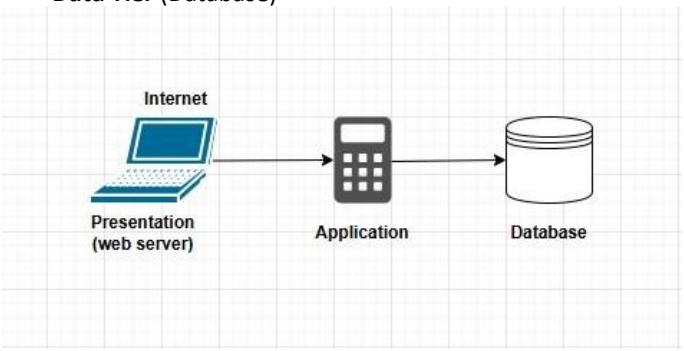
❖ **THEORY EXERCISE:** What is the difference between system software and application software?

Aspect	System Software	Application Software
Purpose	Manages hardware and provides an environment for running applications.	Performs specific tasks or solves specific problems for the user.
Examples	Operating systems (Windows, macOS), device drivers, utilities	Word processors, web browsers, games, media players

Installation	Usually pre-installed; updates are handled automatically.	Installed and updated by the user as needed.
Dependency	Application software depends on system software to function.	Relies on system software to operate.

11. Software Architecture:

- **LAB EXERCISE**: Design a basic three-tier software architecture diagram for a web application
  - **Three-Tier Architecture Overview**:
    - **Presentation Tier** (User Interface)
    - **Application Tier** (Business Logic)
    - **Data Tier** (Database)



- **THEORY EXERCISE:** What is the significance of modularity in software architecture?
  - **Significance of Modularity in Software Architecture:**
    1. Improved Maintainability:
    2. Reusability:
    3. Scalability:
    4. Flexibility and Extensibility:
    5. Separation of Concerns:
    6. Parallel Development:
    7. Testability:
    8. Easier Refactoring:
    9. Enhanced Collaboration:

12. Layers in Software Architecture:

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

- **Introduction:** In this case study, we explore a typical software system with a three-layered architecture: the **Presentation Layer**, the **Business Logic Layer**, and the **Data Access Layer**.
- **Architecture Overview:**
  1. **Presentation Layer**: Deals with the user interface and user interaction.
  2. **Business Logic Layer (BLL)**: Contains the logic and rules of the application, such as calculations, validations, and business processes.
  3. **Data Access Layer (DAL)**: Manages the communication with the database, handling data retrieval, updates, and storage.
- Pre**sentation Layer:**
- **Business Logic Layer (BLL):**
- **Data Access Layer (DAL)**
- **Interaction Between Layers**
- **Conclusion**

❖ **THEORY EXERCISE:** Why are layers important in software architecture?

➢ **Layers Are Important in Software Architecture:**

1. Separation of Concerns
2. Modularity and Maintainability
3. Scalability
4. Flexibility and Extensibility
5. Improved Testability
6. Reusability
7. Simplifies Debugging and Troubleshooting
8. Security and Access Control

13. Software Environments:

❖ **THEORY EXERCISE:** Explain the importance of a development environment in software production

1. Efficiency and Productivity
2. Consistency and Standardization
3. Collaboration and Version Control
4. Debugging and Testing
5. Integration with Other Tools and Systems
6. Environment-Specific Configuration
7. Code Quality and Best Practices
8. Security and Compliance
9. Handling Multiple Environments (Development, Staging, Production

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

➢ **Types of Software Environments:**

- **Development Environment:**
  1. **Source code editing**: Integrated Development Environments (IDEs) such as Visual Studio, PyCharm, or VS Code.
  2. **Version control**: Tools like Git for managing and tracking changes in code.
- **Testing Environment:**
  1. **Test Data**: Mimics real-world data without exposing sensitive information.
  2. **Automated testing tools**: Tools like Selenium, JUnit, or pytest to automate testing.
- **Production Environment:**
  1. **High Availability**: Ensures the system remains up and running 24/7.
  2. **Security**: Implements the necessary security controls to protect user data and application integrity.

➢ **Setting Up a Basic Environment in a Virtual Machine:**

- Install VirtualBox
- Install Ubuntu Linux on the Virtual Machine
- Set Up Development Tools
- Set Up Testing Environment
- Set Up Production Environment (Basic)

14. Source Code:

❖ **THEORY EXERCISE:** What is the difference between source code and machine code?

Aspect	Source Code	Machine Code
Readability	Written in high-level programming languages, human-readable.	Written in binary, not human-readable.
Language	Written in programming languages like Python, Java, C++.	Consists of binary or hexadecimal instructions.
Purpose	Defines the logic and instructions of the software.	Executes directly on the hardware (CPU).
Execution	Cannot be executed directly by a computer	Directly executed by a CPU or processor

15. Github and Introductions:

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

- **Create a GitHub Repository**:
  1. Sign in to GitHub
  2. Create a New Repository
  3. Clone the Repository Locally
- **Commit and Push Code Changes:**
  1. Make Changes to Your Code.
  2. Check Git Status
  3. dd Files to Staging Area.
  4. Commit Your Changes.
  5. Push Your Changes to GitHub.
- **Verifying the Changes on GitHub:**
- **Pulling Updates from GitHub:**
- **Summary of Git Commands:**

❖ **THEORY EXERCISE:** Why is version control important in software development?

- **Track Changes and History**: Version control systems (VCS) allow developers to track all changes made to the codebase over time.
- **Collaboration**: In team-based development, version control facilitates seamless collaboration.
- **Rollback to Previous Versions**: Version control enables developers to revert the code to an earlier, stable version if something goes wrong.
- **Branching and Parallel Development**: Developers can create branches to work on new features, bug fixes, or experiments without affecting the main codebase.
- **Code Review and Quality Assurance**: By using version control, teams can review each other's code, ensuring higher quality and reducing bugs.
- **Backup and Recovery**: The VCS acts as a backup system.

16. Student Account in Github:

❖ **THEORY EXERCISE:** What are the benefits of using Github for students?

1. Learning Industry Standard Tools
2. Access to Open Source Projects
3. Documentation and Project Management
4. Community and Networking
5. Version Control and Collaboration

17. Types of Software:

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

  - **System Software:**
    - **Operating System**:
      1. Windows
      2. Linux
    - **Device Drivers**:
      1. Realtek Audio Drivers
  - **Application Software:**
    - **Productivity Software:** Microsoft Word
    - **Web Browsers:** Google Chrome
    - **Communication Software:**Zoom
  - **Utility Software:**
    - **File Management Tools**: WinRAR
    - **Backup Software**: Google Drive

- **THEORY EXERCISE:** What are the differences between open-source and proprietary software?

Feature	Open-Source Software	Proprietary Software
Source Code	Accessible and modifiable	Closed and inaccessible
Cost	Typically free	Requires purchasing a license
Customization	Highly customizable	Limited customization
Support	Community-based support	Vendor-provided support
Innovation	Fast, community-driven	Controlled by the vendor

18. GIT and GITHUB Training:

- **THEORY EXERCISE:** How does GIT improve collaboration in a software development team?
  1. Distributed Version Control
  2. Branching and Parallel Development
  3. Efficient Collaboration via Pull Requests
  4. Conflict Resolution
  5. Tracking and History
  6. Collaboration Across Teams and Geographies

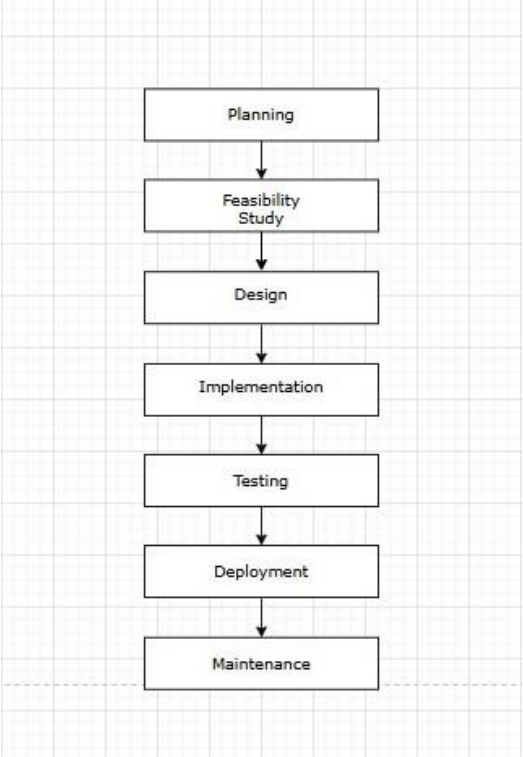
19. Application Software:

- **LAB EXERCISE**: Write a report on the various types of application software and how they improve productivity.
  - **Types of Application Software:**
    - Productivity Software:
    - Communication and Collaboration Software:
    - Creative Software:
    - Database Software:
    - Enterprise Resource Planning (ERP) Software:
  - **How Application Software Improves Productivity:**
    - Automation of Repetitive Tasks**:**
    - Enhanced Collaboration**:**
    - Access to Real-Time Information**:**
    - Improved Organization and Time Management**:**
    - Cost and Resource Savings**:**
    - Better Communication**:**
    - Increased Flexibility**:**

- **THEORY EXERCISE:** What is the role of application software in businesses?
  - Decision Support and Strategic Planning
  - Marketing and Advertising
  - Security and Compliance
  - Human Resource Management (HRM)
  - Customer Relationship Management (CRM)
  - Data Management and Analysis

20. Software Development Process:

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



❖ **THEORY EXERCISE:** What are the main stages of the software development process?

- Planning and Requirement Analysis
- System Design
- Implementation (Coding)
- Testing
- Deployment
- Maintenance
- Documentation

21. Software Requirement:

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

- **Introduction:**
- **Purpose:** The purpose of the **Library Management System** is to provide an automated system that:
  - ✓ Helps manage book inventory.
  - ✓ Allows users to borrow and return books easily.
- **Scope:** The **Library Management System** will handle the following operations:
  - ✓ **Book Management**: Adding, editing, and removing books from the catalog.

- ✓ **User Management**: Allowing users to register, view borrowed books, and track fines.
- **Functional Requirements:**
  - ✓ User Management
  - ✓ Book Management
- N**on-Functional Requirements:**
- **System Architecture:**
- **External Interface Requirements:**
- **Assumptions and Constraints:**

❖ **<u>THEORY EXERCISE:</u>** Why is the requirement analysis phase critical in software development?

- Understanding Client Needs
- Scope Definition
- Avoiding Ambiguity
- Basis for Planning and Estimation
- Quality Assurance
- Communication and Collaboration
- Change Management
- Customer Satisfaction
- Efficient Development Process

22. Software Analysis:

❖ **<u>LAB EXERCISE</u>**: Perform a functional analysis for an online shopping system.

➢ **Functional Analysis for an Online Shopping System:**

- **Introduction:**
- **Functional Requirements:**
  - ✓ User Registration and Authentication.
  - ✓ Product Browsing.
  - ✓ Shopping Cart Management.
  - ✓ Checkout and Order Processing.
  - ✓ Order Tracking and History.
  - ✓ Customer Support and Communication.
  - ✓ Reviews and Ratings.
  - ✓ Discounts and Promotions.
- **Non-Functional Requirements:**
  - ✓ Performance.
  - ✓ Security.
  - ✓ Scalability

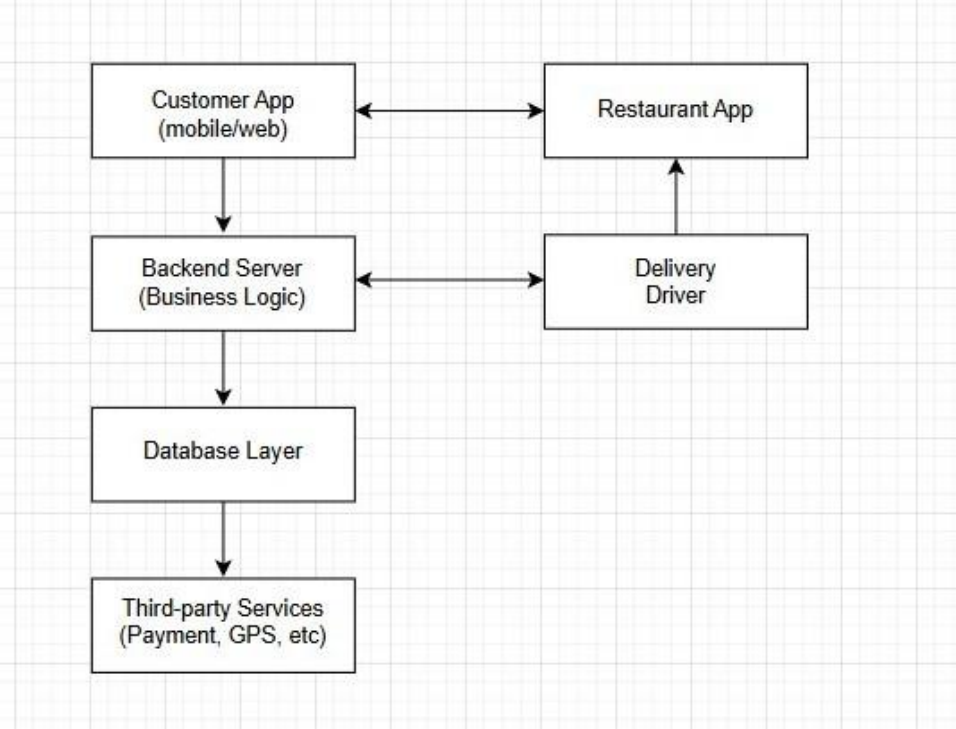
❖ **<u>THEORY EXERCISE:</u>** What is the role of software analysis in the development process?

- Understanding Requirements

- Feasibility Analysis
- System Decomposition
- Data Flow and System Interactions
- Use Cases and User Scenarios
- Business Alignment
- Risk Identification

23. System Design:

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



❖ **THEORY EXERCISE**: What are the key elements of system design?

- **System Architecture:** High-level structure of the system, defining components and interactions.
- **Component Design:** Breakdown of the system into smaller, functional modules.
- **Database Design:** Defining the structure, organization, and access patterns of data
- **User Interface Design:** Design of how users will interact with the system.
- **Security Design:** Strategies to ensure the system is secure from threats.

- **Performance and Scalability:** Ensuring the system can meet performance needs and scale effectively.

.

24. Software Testing:

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

- Test Case: Addition:
  - ✓ **Test Input**: 5 + 3
  - ✓ **Expected Output**: 8
- Test Case: Subtraction:
  - ✓ **Test Input**: 10 – 4
  - ✓ **Expected Output**: 6
- Test Case: Multiplication:
  - ✓ **Test Input**: 4 * 7
  - ✓ **Expected Output**: 28
- Test Case: Division:
  - ✓ **Test Input**: 8 / 2
  - ✓ **Expected Output**: 4

❖ **THEORY EXERCISE:** Why is software testing important?

- Security
- Quality of the product
- Satisfaction of the consumer
- Enhancing the development technique
- Easy even as adding new capabilities
- Determining the performance of the software program

25. Maintenance:

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

➢ **Steps Taken for Critical Maintenance:**

- Grounding of the Fleet:
- Investigation and Discovery:
- Software and Hardware Modifications:
- Testing and Certification:
- Re-certification and Return to Service:

❖ **THEORY EXERCISE:** What types of software maintenance are there?

- **Corrective Maintenance:** Fixing defects, bugs, or errors that occur after deployment.
- **Adaptive Maintenance:** Modifying software to keep it functional with new environments or requirements.
- **Perfective Maintenance:** Enhancing the software by improving performance or adding new features.
- **Preventive Maintenance:** Proactively updating software to avoid future issues.

26. Development:

❖ **THEORY EXERCISE**: What are the key differences between web and desktop applications?

Feature	Web Applications	Desktop Applications
Deployment	Hosted on a server, accessed via browser	Installed on the user's computer
Platform	Platform-independent	Platform-dependent (Windows, macOS, Linux)
Internet Connectivity	Requires internet for most functionality	Can be used offline
Updates	Automatic, server-side updates	Manual updates or auto-updating system
Security	Server-based security, encryption needed	Local security, antivirus protection

27. Web Application:

❖ **THEORY EXERCISE**: What are the advantages of using web applications over desktop applications?

- **Accessibility Anywhere, Anytime:**
  - ✓ **Web applications** are accessible from any device with an internet connection and a web browser (desktop, laptop, tablet, mobile phone).
  - ✓ **Desktop applications**, on the other hand, are typically limited to the device where they are installed, and users must ensure that the software is available on each device they intend to use.
- **Platform Independence:**
  - ✓ **Web applications** are generally platform-independent, meaning they work across different operating systems (Windows, macOS, Linux, etc.) without requiring specific versions for each.

- ✓ **Desktop applications** often require separate versions for each operating system, necessitating additional development work and potentially limiting accessibility for users on different platforms.

- **Automatic Updates:**
  - ✓ **Web applications** are hosted on central servers, so updates can be made quickly and seamlessly for all users. There is no need for users to manually download or install updates.
  - ✓ **Desktop applications** require users to manually download and install updates or rely on the software to auto-update, which can lead to version mismatches or the user using an outdated version.

28. Designing:

❖ **THEORY EXERCISE**: What role does UI/UX design play in application development?

- ➢ **Role of UI Design:**
  - ✓ Visual Appeal:
  - ✓ Brand Identity:
  - ✓ User Interaction:
  - ✓ Consistency:
  - ✓ Accessibility:
- ➢ **Role of UX Design:**
  - ✓ Usability:
  - ✓ Intuitive Flow:
  - ✓ User Research:
  - ✓ Problem Solving:
  - ✓ Interaction Design:

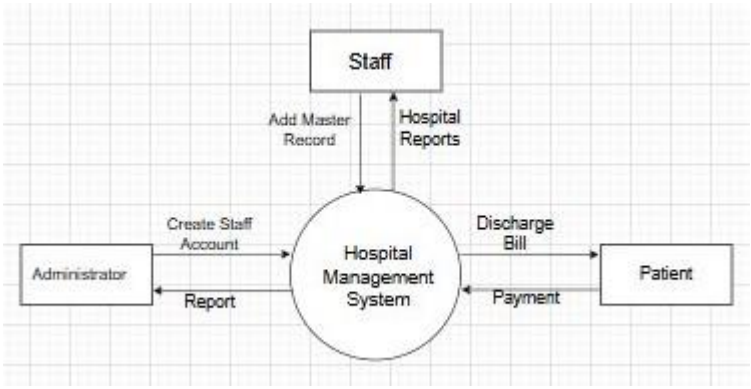
29. Mobile Application:

❖ **THEORY EXERCISE**: What are the differences between native and hybrid mobile apps?

Feature	Native Apps	Hybrid Apps
Development Platform	Platform-specific (iOS/Android)	Web technologies (HTML, CSS, JS)
Performance	High (native code, direct access)	Lower (webview, bridge overhead)
User Experience (UX)	Superior, platform-specific	Less optimal, may feel inconsistent
Cost & Time	Higher cost, longer development	Lower cost, faster development
Maintenance	Separate updates for each platform	Single codebase for all platforms
Access to Device Features	Full access to native features	Limited access, relies on bridge

30. DFD (Data Flow Diagram):

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



- **THEORY EXERCISE**: What is the significance of DFDs in system analysis?
  - Visual Representation of Data Movement
  - Clarifies System Processes:
  - Improves Communication:
  - Identifies Data Sources and Destinations:
  - Supports Problem Identification and System Design:
  - Helps in Requirements Gathering
  - Facilitates System Documentation:
  - Hierarchy and Decomposition

31. Desktop Application:

- **THEORY EXERCISE:** What are the pros and cons of desktop applications compared to web applications?
  - **Desktop Applications:**
    Pros:
    - **Offline Functionality**: Desktop apps can work without an internet connection, making them ideal for environments with limited or no connectivity
    - **Rich User Interface (UI):** Desktop apps can offer a more responsive and sophisticated user interface with better integration with the operating system

- **Security**: Data is stored locally, so users may have more control over their own security and privacy. Additionally, desktop apps may offer tighter integration with local security features like firewalls and antivirus software.

- **Desktop Applications:**
  cons:

  - **Limited Cross-Platform Compatibility:**Desktop apps often need to be developed separately for different operating systems (Windows, macOS, Linux), which can increase development time and costs.
  - **Storage and System Resources:**Desktop apps typically consume local storage and system resources.

- **Web Applications:**
  Pros:

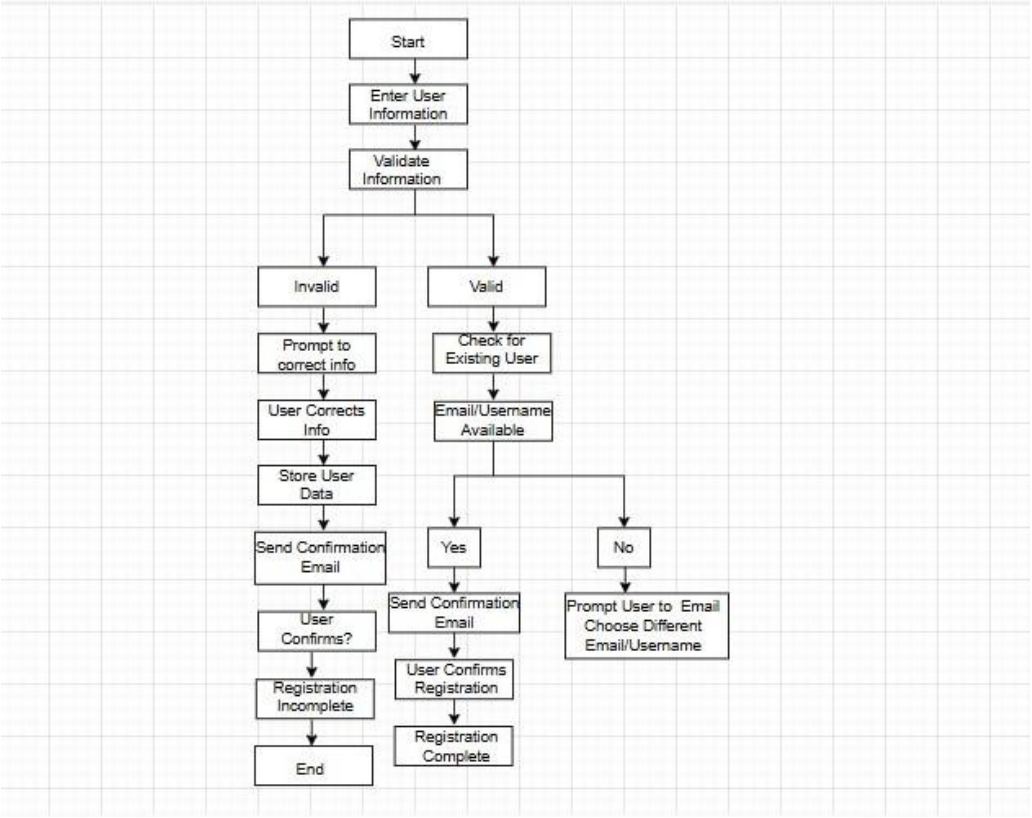
  - **Automatic Updates:**Web applications automatically update in the background without requiring user intervention.
  - **Easy Deployment and Maintenance:** Since web apps are hosted on servers, updates, and maintenance only need to be done in one place (the server), which simplifies management.
  - **Centralized Data Storage:** Data is stored on remote servers, making it easier to backup and secure. Users can access their data from any device with an internet connection.

- **Web Applications:**
  Cons:

  - **imited Access to Device Features**: Web applications are often restricted in accessing local device features, such as file systems, hardware components (e.g., printer, USB devices), and specialized OS functionalities. This can limit their functionality for specific use cases.
  - **Browser Compatibility Issues**: Web apps may face compatibility issues across different web browsers and devices. Ensuring that a web app works seamlessly on all browsers (e.g., Chrome, Firefox, Safari) and devices (smartphones, tablets) may require additional effort.

32. Flow Chart:

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



❖ **<u>THEORY EXERCISE</u>**: How do flowcharts help in programming and system design?

➢ Helping in Programming:

- **Planning and Problem Solving**: Before writing code, flowcharts allow programmers to outline the overall structure and logic of a program.
- **Visualizing Program Logic:** Flowcharts provide a graphical representation of the program's logic, which makes it easier to understand the sequence of operations.
- **Simplifying Complex Processes:** For complex algorithms or programs with multiple decision points and loops, flowcharts break down the complexity.

➢ Helping in System Design:

- **Representing System Workflow:** Flowcharts in system design help visualize the flow of data or processes in a system.
- **Clarifying System Components**: In system design, flowcharts are useful for illustrating how various components interact.
- **Documenting and Standardizing Systems**: Flowcharts are a standard method of documenting system processes, making it easier to manage and communicate the design.