Module 1 – Overview of IT Industry

1. What is a Program?

A program is a set of instructions written in a programming language that tells a computer what to do.

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

    ```
    #include <stdio.h>
    int main() {
       printf("Hello World");
       return 0;
    }
    ```

  - **C++ Program Example (Hello World)**

    ```
    #include <iostream>
    using namespace std;
    int main() {
       cout << "Hello World";
       return 0;
    }
    ```

- **Short Difference (C vs C++)**
  - C uses printf() → procedural language.
  - C++ uses cout → supports both procedural & object-oriented programming.
  - C doesn’t support classes/objects; C++ does.
- **THEORY EXERCISE: What is a program and how it functions?**
  - A program is a collection of instructions that performs a specific task.
  - It functions by taking input, processing it step-by-step, and producing output using the computer’s hardware and software.

2. What is Programming?

Programming is the process of writing instructions, called code, that tell a computer how to perform a specific task or solve a problem.

- **THEORY EXERCISE: Key Steps in the Programming Process**

  1. Define the problem

  2. Plan the solution (algorithm/flowchart)

  3. Write the program (coding)

  4. Compile and run the program

  5. Test and debug errors

  6. Maintain and update the program

3. Types of Programming Languages

- Low-level languages → Machine language, Assembly
- High-level languages → C, C++, Java, Python
- Scripting languages → JavaScript, PHP
- Object-oriented languages → Java, C++, Python
- Procedural languages → C, Pascal
- Markup & Query Languages → HTML, SQL, XML

❖ **THEORY EXERCISE: Difference Between High-level and Low-level Languages (Short Answer)**

- **High-level Languages:**
  - Easy to read and write
  - Closer to human language
  - Portable across systems
  - **Example:** Python, Java, C++
- **Low-level Languages:**
  - Hard to read and understand
  - Closer to machine code
  - Faster execution and more hardware control
  - **Example:** Assembly, Machine code

4. World Wide Web & How Internet Works

- **World Wide Web:**

  The World Wide Web (WWW) is a collection of web pages and resources that can be accessed through the internet using a web browser.

- **How Internet Works:**
  - The internet works by connecting millions of devices through networks.
  - Data travels in small packets through routers and servers until it reaches the destination.

❖ **LAB EXERCISE: Diagram of How Data is Transmitted (Text Diagram)**

```
[Client / Browser]
     |
     | 1. Request (HTTP/HTTPS)
     v
  [Local Router]
     |
     | 2. Sent through ISP
     v
    [ISP]
     |
     | 3. Packets travel through internet
     v
 [Multiple Routers]
     |
     | 4. Reaches Web Server
     v
  [Web Server]
     |
     | 5. Response (HTML/CSS/JS)
     v
[Client / Browser Displays Page]
```

- **THEORY EXERCISE: Roles of Client and Server**

  - **Client Role:**

  The client (web browser) sends a request to access a webpage.

  It receives the response from the server and displays the webpage to the user.

  - **Server Role:**

  The server stores websites, data, and resources.

  It processes the client request and sends back the correct webpage or data.

5. Network Layers on Client and Server

Client and server use network layers (like TCP/IP) to send and receive data.

Each layer performs a specific task such as sending, routing, receiving, and displaying data.

- **TCP (Transmission Control Protocol):** Ensures data is delivered safely
- **IP (Internet Protocol):** Finds the path to deliver the data

❖ **LAB EXERCISE: Simple HTTP Client–Server Program (Example)**

- **Java Server Code (Server.java)**

```
import java.net.*;

import java.io.*;


public class Server {

  public static void main(String[] args) throws Exception {

    ServerSocket ss = new ServerSocket(5000);

    Socket s = ss.accept();


    BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));

    PrintWriter pw = new PrintWriter(s.getOutputStream(), true);


    pw.println("Hello from Server");

    System.out.println("Client: " + br.readLine());


    s.close();

    ss.close();

  }

}
```

- **Java Client Code (Client.java)**

```
import java.net.*;
import java.io.*;

public class Client {
   public static void main(String[] args) throws Exception {
      Socket s = new Socket("localhost", 5000);

      BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
      PrintWriter pw = new PrintWriter(s.getOutputStream(), true);

      System.out.println("Server: " + br.readLine());
      pw.println("Hello Server");

      s.close();
   }
}
```

- **THEORY EXERCISE: TCP/IP Model & Layers**
  - **TCP/IP Model Function:**
    - The TCP/IP model defines how data is sent over the internet.
    - It breaks communication into layers so data can travel reliably from one device to another.
  - **TCP/IP Layers:**

    1. Application Layer

    - Provides services like HTTP, FTP, DNS
    - Used by applications (browser, email)

    2. Transport Layer

    - Ensures reliable communication
    - Uses TCP/UDP protocols

- Breaks data into packets

3. Internet Layer

- Handles IP addressing and routing
- Sends data across different networks

4. Network Access Layer

- Deals with hardware (LAN, Wi-Fi, cables)
- Sends data physically to the next device

6. Client and Server

A client is a device or software that requests services (example: web browser).

A server provides services or resources (example: web server).

- **THEORY EXERCISE: Explain Client–Server Communication**
  - Client–server communication works in a request–response model.
  - The client sends a request to the server, the server processes it, and sends back the response.
  - **Example:** Browser (client) asks for a webpage → Web server sends the page.

7. Types of Internet Connections

- Dial-up
- Broadband (DSL, Cable)
- Fiber-Optic
- Satellite
- Mobile Data (3G/4G/5G)
- Wi-Fi / Wireless

❖ **LAB EXERCISE: Types of Internet Connections – Pros & Cons**

1. Broadband (DSL / Cable)

Pros:

- Fast
- Always ON connection
- Affordable

Cons:

- Speed varies with distance
- Not as fast as fiber
- Can slow down during peak hours

2. Fiber-Optic Internet

Pros:

- Very high speed
- Stable and reliable
- Best for streaming, gaming, heavy work

Cons:

- Expensive
- Not available everywhere

3. Satellite Internet

Pros:

- Available in remote/rural areas
- Wide coverage

Cons:

- Slow speed
- High latency (delay)
- Expensive

4. Mobile Internet (4G/5G)

Pros:

- Portable
- Good speed (5G is very fast)

Cons:

- Depends on network strength
- Data limits

5. Wi-Fi (Wireless Broadband)

Pros:

- Multiple devices connect
- Easy to use

Cons:

- Signal weakens with distance
- Router dependency

❖ **THEORY EXERCISE: Broadband vs Fiber-Optic**

- **Broadband (DSL/Cable):** [Copper cable, medium speed]
  - Uses copper wires
  - Speed is medium
  - Affected by distance and line quality
- **Fiber - Optic Internet:** [Fiber cable, very high speed, more reliable]
  - Uses glass fiber cables
  - Speed is very high (light signals)
  - Not affected by distance
  - More reliable and faster than broadband

8. Protocols

A protocol is a set of rules that defines how computers communicate and exchange data over a network.

Examples: HTTP, HTTPS, FTP, TCP/IP, SMTP.

❖ **LAB EXERCISE: Simulate HTTP & FTP Requests**

Using curl in Command Line

1. **HTTP Request Example:**

```
curl http://example.com
```

- Sends a GET request and shows webpage content.

2. **FTP Request Example:**

```
curl ftp://ftp.example.com --user username:password
```

- Connects to FTP server and retrieves files.

(You can run these in Windows, Linux, or Mac terminal.)

❖ **THEORY EXERCISE: HTTP vs HTTPS**

Feature	**HTTP**	**HTTPS**
Full Form	HyperText Transfer Protocol	HyperText Transfer Protocol Secure
Security	Not secure	Secure (uses SSL/TLS encryption)
Port Number	80	443
Data Protection	No encryption	Encrypted & safe
Use Case	Websites without sensitive data	Online banking, shopping, login pages

9. Application Security

Application security protects software from threats, attacks, and vulnerabilities to ensure safe and reliable operation.

❖ LAB EXERCISE: Three Common Application Security Vulnerabilities

Vulnerability	Explanation	Possible Solution
SQL Injection	Attacker inserts malicious SQL code to access data	Use prepared statements, input validation
Cross-Site Scripting (XSS)	Attacker injects scripts into webpages	Sanitize user input, use Content Security Policy (CSP)
Weak Passwords	Users choose easy-to-guess passwords	Enforce strong passwords, enable multi-factor authentication

❖ THEORY EXERCISE: Role of Encryption

- Encryption converts data into unreadable format using a key.
- Protects sensitive information from unauthorized access.
- Ensures confidentiality, integrity, and secure communication in applications.

In-short answer:

- SQL Injection → Use prepared statements
- XSS → Sanitize input
- Weak Password → Strong passwords + MFA
- Encryption → Keeps data safe from attackers

10. Software Applications and Its Types

Software is a set of programs that tell a computer how to perform tasks.

➢ **Types of Software:**

1. System Software – Helps run the computer (OS, utilities)

2. Application Software – Helps perform specific tasks for users (Word, Browser)

❖ **LAB EXERCISE: Classify 5 Daily Applications**

Application	**Type**
Windows / macOS	System Software
Google Chrome	Application Software
Microsoft Word	Application Software
Antivirus Software	System Software
WhatsApp	Application Software

❖ **THEORY EXERCISE: Difference Between System Software and Application Software**

Feature	**System Software**	**Application Software**
Purpose	Runs and manages computer hardware	Performs specific tasks for users
Examples	OS, Antivirus, Utilities	Word Processor, Browser, Games
User Interaction	Less direct	Direct interaction with users
Dependency	Independent, essential	Depends on system software

11. Software Architecture

Software architecture defines the structure and organization of a software system, showing how components interact to perform tasks efficiently.

❖ **LAB EXERCISE: Three-Tier Software Architecture Diagram**

Text/Diagram Version:

```
┌──────────────────┐
│   Presentation   │ ← User Interface (Browser, App)
│      Layer       │
└────────┬─────────┘
      │
┌────────┴─────────┐
│    Business/     │ ← Application Logic Layer
│     Logic        │
└────────┬─────────┘
      │
┌────────┴─────────┐
│    Data Layer    │ ← Database (MySQL, MongoDB)
└──────────────────┘
```

Explanation:

1. Presentation Layer: Handles UI and user interaction.

2. Business Logic Layer: Processes data and applies rules.

3. Data Layer: Stores and retrieves data from database.

❖ **THEORY EXERCISE: Significance of Modularity**

Modularity means dividing software into separate, independent modules.

Benefits:

1. Easier to develop and maintain

2. Reusable code across applications

3. Helps in testing and debugging

4. Improves clarity and organization of software

12. Layers in Software Architecture

Software architecture is often divided into layers, where each layer has a specific role:

1. Presentation Layer – Handles user interface and interaction.

2. Business Logic Layer – Processes data, applies rules, and performs calculations.

3. Data Access Layer – Manages storage and retrieval of data from databases or files.

❖ **LAB EXERCISE: Case Study Example**

Software System: Online Shopping Application

Layer	**Functionality Example**
Presentation Layer	Web pages or mobile app screens where users browse products, add to cart, and checkout.
Business Logic Layer	Calculates total price, applies discounts, checks inventory, processes orders.
Data Access Layer	Stores product info, user accounts, and order history in the database.

❖ **THEORY EXERCISE: Importance of Layers in Software Architecture**

- **Separation of Concerns:** Each layer has a specific responsibility.
- **Easier Maintenance:** Changes in one layer don't affect others.
- **Reusability:** Modules in layers can be reused across systems.
- **Improved Scalability & Testing:** Layers make software easier to scale and test independently.

13. Software Environments

A software environment is a setup in which software is developed, tested, or deployed.

➢ **Types of Software Environments:**

1. Development Environment – Where software is written and initially tested.

2. Testing Environment – Where software is tested for bugs and performance.

3. Production Environment – Live environment where software is used by end-users.

❖ **LAB EXERCISE: Example Setup**

Task: Set up a basic environment in a virtual machine (VM).

1. Install a VM software (e.g., VirtualBox or VMware).
2. Install an operating system (e.g., Ubuntu).
3. Set up Development Tools:
    - IDE (e.g., Visual Studio Code, Eclipse)
    - Compiler or runtime environment (e.g., Java JDK, Python)
4. Test by writing and running a simple program.

❖ **THEORY EXERCISE: Importance of Development Environment**

- Provides a safe workspace for developers.
- Helps in writing, debugging, and testing code before release.
- Ensures consistency and reproducibility across development teams.
- Reduces errors in production by catching bugs early.

14. Source Code

Source code is a set of instructions written by a programmer in a programming language (e.g., C, Java, Python).

- **LAB EXERCISE: Write and Upload Source Code to GitHub**

C Source Code Example (“Hello World”):

```
#include <stdio.h>

int main() {

   printf("Hello World\n");

   return 0;

}
```

- **Steps to Upload to GitHub:**

1. Create a GitHub repository.

2. Save your source code file (e.g., hello.py) locally.

3. Open terminal/command prompt:

```
git init

git add hello.py

git commit -m "First commit"

git branch -M main

git remote add origin <repository_url>

git push -u origin main
```

4. Check your file uploaded in GitHub repository.

- **THEORY EXERCISE: Difference Between Source Code and Machine Code**

Feature	**Source Code**	**Machine Code**
Written in	High-level programming language	Binary code (0s and 1s)
Readable by	Humans	Computers only
Example	print("Hello World")	10110000 01101100 …
Purpose	To instruct computer logically	Executable code for the computer

15. GitHub and Introduction

GitHub is an online platform for hosting, managing, and collaborating on code using Git version control.

- **LAB EXERCISE: Create a GitHub Repository and Commit/Push Code**

  Steps:

  1. Go to GitHub → Click New Repository → Give it a name → Create.
  2. Open terminal/command prompt on your local machine.
  3. Navigate to your project folder: cd path/to/your/project
  4. Initialize Git: git init
  5. Add your code files: git add filename
  6. Commit changes: git commit -m "Initial commit"
  7. Link to GitHub repository: git remote add origin <repository_url>
  8. Push code to GitHub: git push -u origin main

  Now your code is on GitHub.

- **THEORY EXERCISE: Importance of Version Control**
  - Tracks changes to code over time.
  - Allows multiple developers to collaborate safely.
  - Makes it easy to revert to previous versions if errors occur.
  - Helps manage releases and maintain code history.

In-short answer:

GitHub: Online code repository

Commit/Push: Save and upload changes

Version Control: Track changes, collaborate, and revert errors

16. Student Account in GitHub

❖ **LAB EXERCISE: Create a Student Account and Collaborate on a Project**

Steps to Create a Student GitHub Account

1. Go to github.com

2. Click Sign Up

3. Enter email → Create password → Choose username

4. Verify email

5. (Optional but useful) Apply for GitHub Student Developer Pack for free tools.

Collaborate on a Small Project

1. One student creates a repository.

2. Go to Settings → Manage Access → Invite collaborator.

3. Classmate accepts the invite.

4. Both can:

   - Clone the repository
   - Create/update files
   - Commit and push changes
   - Work together using pull requests

Example commands:

```
git clone <repo_url>

git add .

git commit -m "Added new file"

git push
```

❖ **THEORY EXERCISE: Benefits of GitHub for Students**

- Helps build a portfolio to show skills to employers.
- Teaches real-world version control (Git), used by software companies.
- Allows team collaboration on projects.
- Helps track all changes with history and backup.
- Provides access to open-source projects and learning resources.
- GitHub Student Developer Pack gives free tools like hosting, cloud services, IDEs, etc.

17. Types of Software

- **LAB EXERCISE: List and Classification of Software You Use**

Software Name	**Category**
Android OS	System Software
Windows/Linux	System Software
WhatsApp	Application Software
Chrome Browser	Application Software
MS Word	Application Software
Antivirus (Avast)	Utility Software
WinRAR/7-Zip	Utility Software
Disk Cleanup Tool	Utility Software

- **THEORY EXERCISE: Difference Between Open-source and Proprietary Software**

Open-source Software

- Source code is publicly available.
- Anyone can view, modify, and distribute it.
- Usually free.
- Examples: Linux, Firefox, VLC.

Proprietary Software

- Source code is closed and controlled by a company.
- Users cannot modify it.
- Usually paid or licensed.
- Examples: Windows, MS Office, Photoshop.

18. GIT and GitHub Training

- **LAB EXERCISE: Practice Cloning, Branching, and Merging**

1. **Clone a Repository**

```
git clone <repository_url>
```

2. **Create a New Branch**

```
git checkout -b feature-branch
```

3. **Make Changes and Commit**

```
git add .
git commit -m "Updated feature"
```

4. **Merge Branch into Main**

```
git checkout main
git merge feature-branch
```

This helps you learn how developers work with versions of code safely.

- **THEORY EXERCISE: How GIT Improves Collaboration**

Git improves teamwork by:

- Tracking every change made by every developer.
- Allowing multiple developers to work on the same project without overwriting each other's code.
- Using branches so everyone can work on features independently.
- Helping merge work together safely and efficiently.
- Providing a full history of changes, making it easy to fix mistakes.
- Supporting platforms like GitHub for remote collaboration.

In-short answer:

Git improves collaboration by allowing multiple developers to work on the same project using branches, tracking changes, merging updates safely, and keeping a complete history of code changes.

19. Application Software

Application software is software that helps users perform specific tasks such as writing documents, browsing the internet, or sending messages.

❖ **LAB EXERCISE: Report on Types of Application Software & How They Improve Productivity**

➢ **Types of Application Software:**

1. **Word Processing Software**
   - Example: MS Word, Google Docs
   - Helps create documents, reports, and letters quickly.
2. **Spreadsheet Software**
   - Example: MS Excel
   - Helps calculate, analyze data, create graphs → improves accuracy and speed.
3. **Presentation Software**
   - Example: PowerPoint
   - Helps create slides for teaching, business meetings, and explanations.
4. **Database Management Software**
   - Example: MySQL, MS Access
   - Helps store and manage large data easily.
5. **Web Browsers**
   - Example: Chrome, Firefox
   - Help access information and online applications.
6. **Communication Software**
   - Example: WhatsApp, Gmail
   - Helps in fast communication and collaboration.
7. **Media Players & Editing Software**
   - Example: VLC, Photoshop
   - Helps create/edit media files, improving creative productivity.

➢ **How They Improve Productivity**

- Automate tasks (typing, calculations, reports)
- Save time and reduce manual work
- Improve communication and collaboration
- Increase accuracy in data handling

  - Make learning and presentations easier

- **THEORY EXERCISE: Role of Application Software in Businesses**

  Application software helps businesses by:

  - Improving efficiency (faster calculations, data management)
  - Enhancing communication between teams and customers
  - Supporting decision-making with data analysis tools
  - Automating tasks like billing, inventory, payroll
  - Managing customer relationships (CRM software)
  - Creating documents, reports, presentations for business operations

In-short Answer:

Application software helps businesses by increasing productivity, improving communication, automating tasks, managing data, and supporting decision-making.

20. Software Development Process

The software development process is a step-by-step method to plan, create, test, and maintain software from start to finish.

- **LAB EXERCISE: Flowchart of SDLC**

```
┌───────────────────┐
|   1. Requirement   |
|      Analysis      |
└───────┬───────────┘
        |
┌───────▼───────────┐
|     2. Design      |
|       (Architecture)  |
└───────┬───────────┘
        |
┌───────▼───────────┐
|   3. Development   |
|     (Coding)       |
└───────┬───────────┘
        |
┌───────▼───────────┐
|     4. Testing     |
|       (Find bugs)  |
└───────┬───────────┘
        |
┌───────▼───────────┐
|    5. Deployment   |
|      (Release)     |
└───────┬───────────┘
        |
┌───────▼───────────┐
```

```
|     6. Maintenance      |
|       (Updates/Fixes)  |
└─────────────────────────┘
```

This is the SDLC Flowchart.

- **THEORY EXERCISE: Main Stages of the Software Development Process**

The main stages of SDLC are:

1. Requirement Analysis

Understand what the client needs.

2. Design

Create architecture, UI design, database design.

3. Development (Coding)

Programmers write the actual code.

4. Testing

Software is tested to find and fix bugs.

5. Deployment

Software is released to users.

6. Maintenance

Updating, fixing issues, adding new features after release.

In-short Answer:

SDLC includes Requirement Analysis, Design, Development, Testing, Deployment, and Maintenance.

21. Software Requirement

Software requirements describe what a software system should do and how it should work.

- **LAB EXERCISE: Requirement Specification for a Simple Library Management System**

  1. Functional Requirements (What the system should do):

  - The system should allow adding new books.
  - The system should allow issuing and returning books.
  - The system should store student/member details.
  - The system should search books by title, author, or ID.
  - The system should show available and issued books.

  2. Non-Functional Requirements (How the system should behave):

  - The system should be easy to use.
  - The system should work fast and respond quickly.
  - Data should be stored securely.
  - System should be available 24/7.

- **THEORY EXERCISE: Why Requirement Analysis Is Critical**

  Meaning:

  Requirement analysis means understanding what the user needs before building the software.

  Why it is critical:

  - Ensures the software meets user expectations.
  - Prevents mistakes and misunderstandings early.
  - Saves time and cost in development.
  - Helps developers clearly understand what to build.
  - Reduces the chances of rework and project failure.

22. Software Analysis

Software analysis means studying and understanding what a software system should do, what features it needs, and how it should work.

- **LAB EXERCISE: Functional Analysis for an Online Shopping System**

  Functional Analysis (What the system should do):

  1. **User Management**
     - Users can register and log in.
     - Users can update their profile.
  2. **Product Management**
     - Users can view product categories.
     - Users can search and filter products.
     - Admin can add, update, and delete products.
  3. **Shopping Cart**
     - Users can add or remove items from the cart.
     - Users can view total price.
  4. **Order Processing**
     - Users can place orders.
     - System generates order ID and confirms purchase.
     - Users can track order status.
  5. **Payment**
     - Supports online payments (UPI, card, net banking).
     - Shows payment success or failure messages.
  6. **Notifications**
     - Sends email/SMS alerts for order confirmation and delivery updates.

- **THEORY EXERCISE: Role of Software Analysis in Development**

  Software analysis helps understand what the software must do and what features are needed.

- **Role in Development:**
  - Identifies user needs and system requirements.
  - Ensures developers know exactly what to build.
  - Reduces mistakes and confusion during development.
  - Helps in planning system design and features.

- Saves time and cost by avoiding rework.
- Acts as the foundation for the entire SDLC process.

In-short answer:

Software analysis defines user needs and system requirements, ensuring the right software is built correctly.

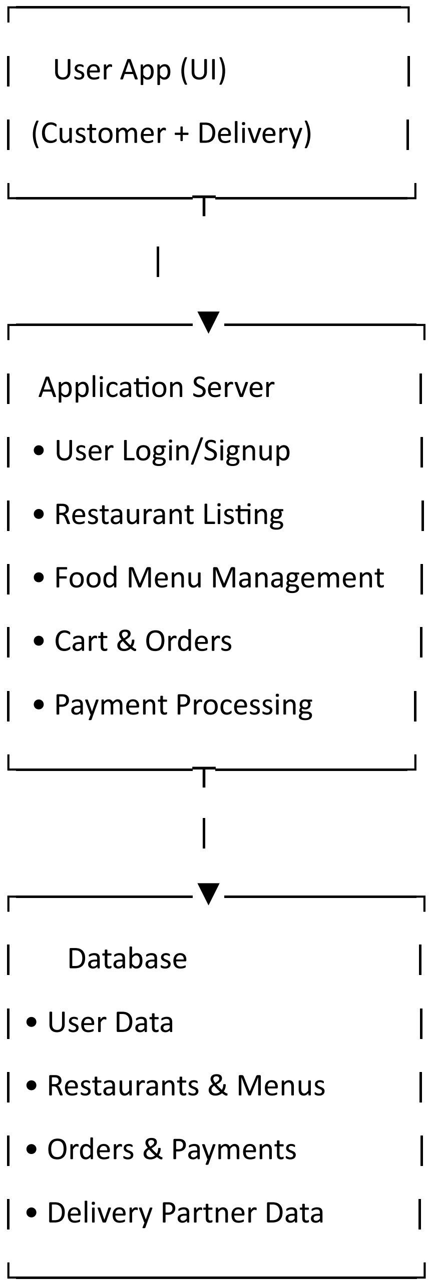
23. System Design

System design means planning and creating the structure of a software system before coding.

System design defines the architecture, components, data flow, UI, and security of a software system to ensure it works efficiently and meets requirements.

❖ LAB EXERCISE: Basic System Architecture for a Food Delivery App

System Architecture (Simple Block Diagram)



User App (UI)

(Customer + Delivery)

Application Server

- User Login/Signup
- Restaurant Listing
- Food Menu Management
- Cart & Orders
- Payment Processing

Database

- User Data
- Restaurants & Menus
- Orders & Payments
- Delivery Partner Data

- **Main Components:**
  - **Frontend (Mobile App):** User interface
  - **Backend Server:** Business logic
  - **Database:** Stores user, order, and restaurant data

- **THEORY EXERCISE: Key Elements of System Design**

  The key elements of system design are:

  1. Architecture Design – structure of the whole system

  2. User Interface Design – how the user will interact with the software

  3. Database Design – how data will be stored and organized

  4. Component/Module Design – dividing the system into separate parts

  5. Data Flow Design – how data moves between components

  6. Security Design – protecting data and users

  7. Integration Design – how different modules connect and work together

24. Software Testing

Software testing is the process of finding and fixing errors to ensure the software is reliable and works correctly.

- ❖ **LAB EXERCISE: Test Cases for a Simple Calculator Program**

Test Case No.	Operation	Input	Expected Output	Result (Pass/Fail)
1	Addition	5 + 3	8	Pass
2	Subtraction	10 - 4	6	Pass
3	Multiplication	6 × 7	42	Pass
4	Division	20 ÷ 5	4	Pass
5	Division by 0	5 ÷ 0	Error message	Pass

- ❖ **THEORY EXERCISE: Why Software Testing is Important**

  - Ensures software works correctly and meets requirements.
  - Detects bugs and errors before release.
  - Improves software quality and reliability.
  - Reduces maintenance cost and user complaints.
  - Builds user confidence in the software.

25. Maintenance

Software maintenance is the process of updating, fixing, and improving software after it has been delivered to users.

❖ **LAB EXERCISE: Real-World Case of Critical Maintenance**

Case Example: Facebook Platform Outage

- In 2021, Facebook, Instagram, and WhatsApp faced a major global outage.
- Engineers had to perform critical maintenance to fix server configuration errors and restore services.
- Maintenance included updating network settings, debugging server issues, and restoring data connections.
- Result: Services were restored, preventing long-term business and user impact.

❖ **THEORY EXERCISE: Types of Software Maintenance**

1. Corrective Maintenance – Fixing bugs and errors.

2. Adaptive Maintenance – Updating software to work with new hardware, OS, or technologies.

3. Perfective Maintenance – Improving performance, adding new features, or optimizing software.

4. Preventive Maintenance – Preventing future issues by monitoring and updating code.

In-short answer:

- Corrective: Fix bugs
- Adaptive: Adjust to new environment
- Perfective: Improve features/performance
- Preventive: Avoid future problems

26. Development

Development means the process of writing, creating, and building software programs to meet user requirements.

- **THEORY EXERCISE: Web vs Desktop Applications**

Feature	**Web Applications**	**Desktop Applications**
Installation	Runs in a web browser; no installation needed	Must be installed on a specific device
Accessibility	Can be accessed from any device with internet	Limited to the device it is installed on
Updates	Updated on server; users get latest version automatically	Updates must be installed manually on each device
Performance	Depends on internet speed and server	Usually faster; runs locally on device
Examples	Gmail, Facebook, Google Docs	MS Word, Photoshop, VLC Player

27. Web Application

A web application is a software program that runs in a web browser and can be accessed over the internet.

Examples: Gmail, Google Docs, Facebook, Amazon.

Advantages of Web Applications over Desktop Applications

Advantage	**Explanation**
Accessibility	Can be used from any device with an internet connection.
No Installation Needed	Users don't need to install the app on their device.
Automatic Updates	Updates are applied on the server; users always get the latest version.
Cross-Platform Compatibility	Works on different devices (PC, tablet, smartphone) with a browser.
Lower Maintenance Cost	Easier for developers to maintain one version on the server.

28. Designing

Designing – UI/UX Role in Application Development

UI (User Interface) Design:

- Focuses on how the app looks (layout, colors, buttons, fonts).
- Makes the application visually appealing and easy to navigate.

UX (User Experience) Design:

- Focuses on how the app feels and how users interact with it.
- Ensures the application is efficient, intuitive, and user-friendly.

❖ **THEORY EXERCISE: UI/UX Role in Application Development**

  - UI/UX design plays a crucial role in application development by ensuring the app is user-friendly, visually appealing, and easy to navigate.
  - It improves user satisfaction, reduces errors, and increases adoption and engagement with the application.

29. Mobile Application

A mobile application is a software program designed to run on smartphones and tablets to perform specific tasks.

Examples: WhatsApp, Instagram, Google Maps.

❖ **THEORY EXERCISE: Differences between native and hybrid mobile apps**

Feature	**Native Mobile Apps**	**Hybrid Mobile Apps**
Development	Built for a specific platform (iOS or Android)	Built using web technologies (HTML, CSS, JavaScript) and wrapped for multiple platforms
Performance	High performance; uses device's full capabilities	Slightly lower performance; depends on web view
Installation	Installed from App Store/Play Store	Installed like native apps but code runs inside a web view
Access to Device Features	Full access (camera, GPS, sensors)	Limited access; may require plugins for some features
Development Cost	Higher; separate code for each platform	Lower; single code works on multiple platforms
Examples	Instagram (iOS/Android versions)	Instagram Lite, Ionic apps

In-short answer:

Native apps: High-performance, platform-specific, full device access.

Hybrid apps: Cross-platform, lower cost, uses web technologies, limited device access.

30. DFD (Data Flow Diagram)

A Data Flow Diagram (DFD) is a graphical representation of how data moves through a system.

It shows processes, data stores, inputs, and outputs clearly.

In short: DFD visualizes the flow of information in a system.

❖ **LAB EXERCISE: DFD for a Hospital Management System**

Level 0: (Context Diagram – Simple)

```
┌──────────────┐
|   Patient    |
└──────┬──────┘
       |
┌──────▼──────┐
|   Hospital   |
|  Management  |
|    System    |
└──────┬───────┘
       |
┌──────▼───────┐
|    Doctor    |
└──────────────┘
```

Level 1: DFD (Detailed Processes)

- **Processes:**

  1. Register Patient

  2. Schedule Appointment

  3. Manage Medical Records

  4. Billing and Payments

  5. Generate Reports

- **Data Stores:** Patient DB, Doctor DB, Billing DB

- **External Entities:** Patient, Doctor, Staff, Insurance

- **THEORY EXERCISE: Significance of DFDs in System Analysis**
  - Helps understand system processes and data flow easily.
  - Provides a visual representation for both developers and clients.
  - Identifies redundancies and bottlenecks in data flow.
  - Acts as a foundation for system design and documentation.
  - Helps in communicating system functionality to stakeholders.

31. Desktop Application

A desktop application is software that is installed and runs directly on a computer (like Windows, macOS, or Linux) instead of in a web browser.

Examples: MS Word, VLC Player, Calculator, Photoshop.

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

  Java GUI Calculator Example: (Addition, Subtraction, Multiplication, Division)

```
import javax.swing.*;

import java.awt.event.*;


public class ShortCalc {

  public static void main(String[] args) {

    JFrame f = new JFrame("Calculator");


    JTextField n1 = new JTextField();

    JTextField n2 = new JTextField();

    JTextField ans = new JTextField();


    JButton add = new JButton("+");

    JButton sub = new JButton("-");

    JButton mul = new JButton("*");

    JButton div = new JButton("/");


    n1.setBounds(20,20,150,30);

    n2.setBounds(20,60,150,30);

    add.setBounds(20,100,50,30);

    sub.setBounds(80,100,50,30);

    mul.setBounds(140,100,50,30);

    div.setBounds(200,100,50,30);

    ans.setBounds(20,150,230,30);
```

```
        ActionListener act = e -> {
            double a = Double.parseDouble(n1.getText());
            double b = Double.parseDouble(n2.getText());
            double r = 0;

            if(e.getSource()==add) r = a + b;
            if(e.getSource()==sub) r = a - b;
            if(e.getSource()==mul) r = a * b;
            if(e.getSource()==div) r = a / b;

            ans.setText("Result: " + r);
        };

        add.addActionListener(act);
        sub.addActionListener(act);
        mul.addActionListener(act);
        div.addActionListener(act);

        f.add(n1); f.add(n2); f.add(add); f.add(sub);
        f.add(mul); f.add(div); f.add(ans);

        f.setSize(300,250);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

❖ **THEORY EXERCISE: What are the pros and cons of Desktop applications compared to Web applications?**

- **Pros of Desktop Applications**

  1. Faster performance – Runs directly on the computer.
  2. Works offline – No internet needed.
  3. Better hardware access – Can use system resources efficiently.
  4. More stable – Not affected by browser or network issues.

- **Cons of Desktop Applications**

  1. Requires installation on each device.
  2. Platform dependent – Different versions needed for Windows, macOS, Linux.
  3. Manual updates required by users.
  4. Limited accessibility – Can only be used on the device where installed.

32. Flowchart

A flowchart is a diagram that shows the step-by-step sequence of operations in a process or program using symbols and arrows.

In short: Flowcharts visually represent the logic of a system or program.

❖ **LAB EXERCISE: Flowchart for Basic Online Registration System**

- **Flowchart Steps:**

  1. Start
  2. Enter user details (Name, Email, Password)
  3. Validate input

     If invalid → Show error → Go back to step 2

     If valid → Proceed

  4. Save details to database
  5. Display "Registration Successful" message
  6. End

❖ **THEORY EXERCISE: How Flowcharts Help**

- Helps visualize the logic before coding.
- Makes it easier to understand complex processes.
- Helps identify errors or bottlenecks in the system design.
- Serves as a communication tool between developers and stakeholders.
- Simplifies documentation of programs and systems.

In-short answer:

Flowcharts show program logic visually, helping in planning, understanding, and designing software efficiently.