Java – Full stack Assignment 2024(Practical Questions)

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

➔

```
#include<stdio.h>
void main()
{
Printf(“hello world”);
}
```

And

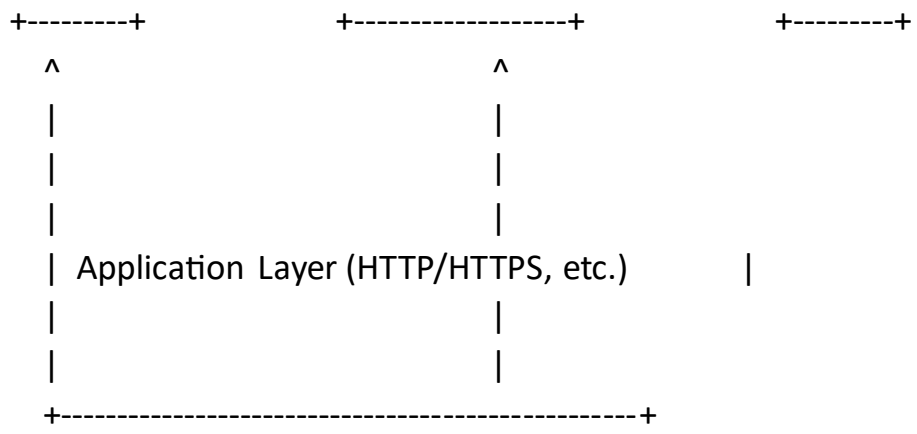
```
#include <iostream>

int main() {
   std::cout << "Hello World";
   return 0;
}
```

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

-> Data transmission from a client to a server over the internet follows the client-server model. A client, typically a web browser or email client, initiates a request to a server, which then processes the request and sends back a response. This communication relies on networking protocols like HTTP/HTTPS, TCP/IP, or WebSocket.

```
+---------+              +------------------+              +---------+
| Client  | -> Network Request | Internet       | -> Response | Server |
| (e.g.,  |              | (Routers, etc.)  |              | (e.g.,  |
| Browser) | ---------- | ---------- | ---------- | Web Server) |
```



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

->Here's a simple example of HTTP client-server communication written in Python, using built-in libraries:

Server Code (Python HTTP Server)

```
from http.server import BaseHTTPRequestHandler, HTTPServer
class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        # Respond with a 200 OK status
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        # Send a response message
        self.wfile.write(b'Hello from the server!')
def run(server_class=HTTPServer,
handler_class=SimpleHTTPRequestHandler):
    server_address = ('localhost', 8080)
    httpd = server_class(server_address, handler_class)
    print('Starting server on http://localhost:8080')
    httpd.serve_forever()
if _name_ == '_main_':
    run()
```

Client Code (Python HTTP Client)

```
import requests
def fetch_data():
    url = 'http://localhost:8080'
```

```
    response = requests.get(url)
    print('Status Code:', response.status_code)
    print('Response Text:', response.text)
if _name_ == '_main_':
    fetch_data()
```

How to Run It

1. Start the server:
   - Save the server code in a file named server.py.
   - Run it: python server.py
2. Run the client:
   - Save the client code in a file named client.py.
   - Run it: python client.py

You'll see the message "Hello from the server!" printed by the client, showing successful HTTP communication.

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

-> Here's a researched comparison of **different types of internet connections**, including **Broadband (DSL, Cable), Fiber, Satellite, and Mobile Data (4G/5G)**, along with their **pros and cons**:

---

1. DSL (Digital Subscriber Line)

Uses existing telephone lines.

Pros:

- Widely available in urban and rural areas.
- Does not interfere with phone usage.
- Affordable.

Cons:

- Slower speeds compared to fiber or cable.
- Speed decreases with distance from provider.

---

2. Cable Internet

Delivered through coaxial cable TV lines.

Pros:

- Faster than DSL.

- Reliale connection.
- Good for streaming and gaming.

Cons:

- Speed may reduce during peak hours (shared bandwidth).
- More expensive than DSL.

---

3. Fiber Optic Internet

Uses light signals through glass cables.

Pros:

- Very high speed (up to 1 Gbps or more).
- Extremely reliable and low latency.
- Best for heavy data use, gaming, streaming, remote work.

Cons:

- Limited availability (mostly urban areas).
- Expensive to install.

---

4. Satellite Internet

Transmits data via satellites orbiting Earth.

Pros:

- Available almost anywhere on Earth.
- Good option for remote/rural areas.

Cons:

- High latency (delay), not ideal for gaming/video calls.
- Affected by weather.
- Lower speeds and high cost.

---

5. Mobile Internet (4G/5G)

Delivered via cellular networks.

Pros:

- Portable and flexible.
- 5G offers very high speeds and low latency.
- Easy setup, no cables required.

Cons:

- Coverage varies by region.
- Speed depends on signal strength.
- Data caps and throttling are common

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

-> Here's how you can **simulate HTTP and FTP requests using curl** from the **command line**.

Simulating HTTP Requests with curl

1. GET Request (HTTP)

bash

CopyEdit

curl http://example.com

Fetches the HTML content of the webpage.

---

2. POST Request (HTTP)

bash

CopyEdit

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

Sends form data (username and password) as a POST request.

---

3. GET Request with Headers

bash

CopyEdit

curl -H "Accept: application/json" http://example.com/api/data

Adds custom headers to the request.

---

4. Download a File

bash

CopyEdit

curl -O http://example.com/file.zip

Saves the file from the server to your local directory.

---

5. View Response Headers

bash

CopyEdit

curl -I http://example.com

Shows only the HTTP response headers.

Simulating FTP Requests with curl

1. Download a File (Anonymous FTP)

bash

CopyEdit

curl ftp://speedtest.tele2.net/1MB.zip -O

---

2. Download a File (FTP with Credentials)

bash

CopyEdit

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

Logs in with a username and password to download the file.

---

3. Upload a File to FTP Server

bash

CopyEdit

curl -T myfile.txt -u username:password ftp://ftp.example.com/uploads/

Uploads myfile.txt to the specified FTP folder.

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

-> Five common applications used daily can be classified as either system software or application software. System software includes the operating system (like Windows or macOS) and device drivers, which manage the computer's hardware and resources. Application software includes web browsers, word processors, media players, and communication tools like Slack, which are used for specific tasks by the user.

Here's a breakdown of the classification:

1. Operating System (System Software):

This is the core software that manages all hardware and software resources of a computer, like Windows, macOS, or Linux. It's essential for the computer to function.

2. Web Browser (Application Software):

Applications like Chrome, Firefox, or Safari allow users to access and interact with websites and online content. They are not essential for basic computer operation, but very commonly used.

3. Word Processor (Application Software):

Programs like Microsoft Word or Google Docs enable users to create, edit, and format text documents. They are used for various tasks related to text-based content.

4. Media Player (Application Software):

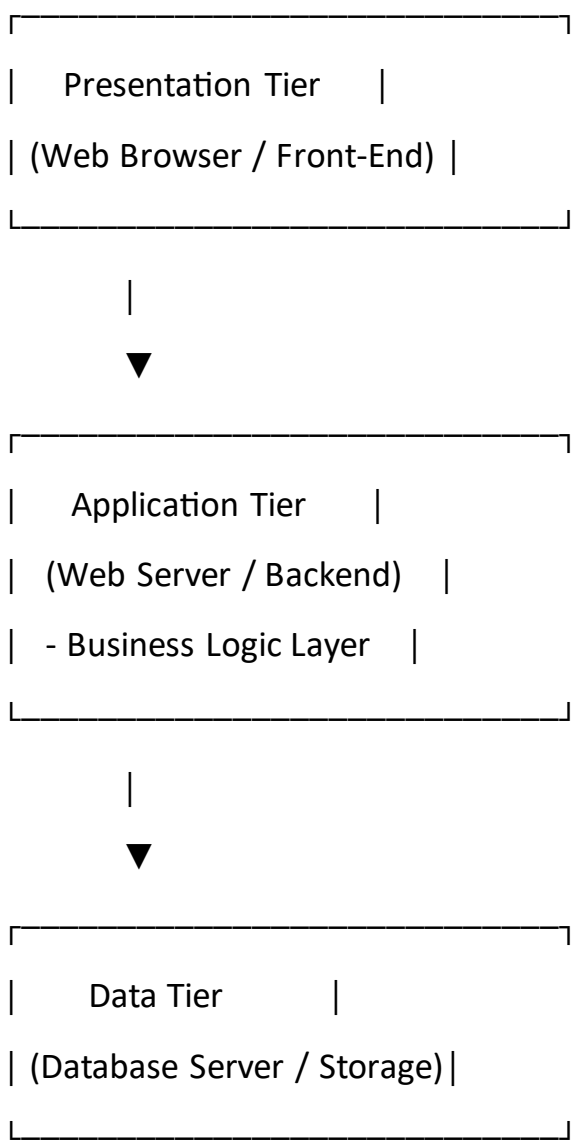
Applications such as VLC or Windows Media Player allow users to play audio and video files. They are used for multimedia consumption.

5. Communication Tool (Application Software):

Applications like Slack or Microsoft Teams are used for real-time communication and collaboration among teams. They facilitate messaging, video calls, and file sharing.

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

-> Here's a basic three-tier software architecture diagram for a typical web application with explanations:



1. Presentation Tier (Client-side / UI)

- **Role**: Interacts with the user.
- **Tech examples**: HTML, CSS, JavaScript, React, Angular, etc.
- **Example component**: Web browser.

---

2. Application Tier (Server-side Logic)

- **Role**: Processes user input, runs business logic, communicates between UI and database.
- **Tech examples**: Node.js, Django, Flask, Spring Boot, etc.
- **Example component**: Web server running backend API.

---

3. Data Tier (Storage Layer)

- **Role**: Manages and stores application data.
- **Tech examples**: MySQL, PostgreSQL, MongoDB, Oracle, etc.
- **Example component**: Database server.

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

-> **Architecture Breakdown (Three-Tier System):**

---

1. Presentation Layer (User Interface)

Purpose: Interacts with the user.

Technologies Used:

- HTML, CSS, JavaScript
- React (Frontend Framework)
- Bootstrap (Styling)

Functionality:

- Displays a list of available books.
- Provides search and filter options.
- Lets users add books to the cart and place orders.
- Shows login/register pages.

Example:

html

CopyEdit

```
<!-- Book display card -->
<div class="book-card">
  <h3>The Alchemist</h3>
  <p>By Paulo Coelho</p>
  <button>Add to Cart</button>
</div>
```

---

2. Business Logic Layer (Application Layer)

Purpose: Processes data and handles logic.

Technologies Used:

- Node.js with Express (API Server)
- Python with Django (Alternative backend)

Functionality:

- Validates user credentials on login.
- Calculates total order value including tax and discounts.
- Checks inventory before confirming orders.
- Manages user sessions and roles (admin, customer).

Example:

javascript

CopyEdit

```
// Pseudocode: Add book to cart
function addToCart(userId, bookId) {
  if (bookInStock(bookId)) {
    cartService.add(userId, bookId);
    return "Book added!";
```

```
  } else {
    return "Out of stock";
  }
}
```

---

3. Data Access Layer (Database Layer)

Purpose: Manages interaction with the database.

Technologies Used:

- MySQL / MongoDB
- Sequelize / Mongoose (ORM/ODM)

Functionality:

- Retrieves book data from the database.
- Stores customer orders and payment history.
- Updates stock quantity after purchase.
- Stores user profile and authentication data.

Example (SQL):

sql

CopyEdit

```
SELECT * FROM books WHERE title LIKE '%alchemist%';
```

Example (MongoDB):

javascript

CopyEdit

```
Book.find({ title: /alchemist/i });
```

---

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

-> **Types of Software Environments**

Software environments are used at various stages of the **Software Development Life Cycle (SDLC)** to ensure that applications work reliably before being deployed to users.

1. Development Environment

- **Purpose**: Where developers write and test code.
- **Tools**: Code editors (VS Code, Eclipse), compilers, debuggers.
- **Traits**:
    - Frequent code changes.
    - Includes mock data or sample inputs.
    - May include local databases.
- **Example**: A developer's laptop running a local web server with database.

---

2. Testing Environment

- **Purpose**: To test functionality, performance, and security.
- **Tools**: Selenium, JUnit, Postman, JMeter, etc.
- **Traits**:
    - Closer to production setup.
    - Used by QA teams.
    - Contains real or mock data.
- **Types of Testing**: Unit testing, integration testing, system testing, UAT.

---

3. Production Environment

- **Purpose**: The live environment accessed by end users.
- **Traits**:
    - Fully optimized and secure.

- No experimental code.
- Requires backups, logging, and monitoring.

- **Tools**: Web servers (Nginx, Apache), monitoring tools (Prometheus, Grafana).

Part 2: Set Up a Basic Environment in a Virtual Machine (VM)

Let's create a **basic development environment** inside a virtual machine using **Ubuntu Server** and install necessary tools.

Prerequisites:

- VirtualBox (or VMware)
- Ubuntu ISO (Ubuntu Server or Desktop)

Steps to Set Up the Environment

Step 1: Install the Virtual Machine

1. Open **VirtualBox**.
2. Create a new VM (e.g., "DevEnv").
3. Allocate RAM (2 GB recommended).
4. Create a virtual hard disk (10+ GB).
5. Mount the Ubuntu ISO and start installation.

Step 2: Set Up a Development Stack (e.g., LAMP or Node.js)

For a Node.js environment:

bash

CopyEdit

sudo apt update

sudo apt install nodejs npm -y

node -v

npm -v

For a LAMP stack (Linux, Apache, MySQL, PHP):

bash

CopyEdit

sudo apt update

sudo apt install apache2 mysql-server php libapache2-mod-php -y

sudo systemctl start apache2

sudo systemctl enable apache2

Visit http://localhost from the VM browser or use port forwarding to access it from your host.

Step 3: Create a Sample Web App

bash

CopyEdit

mkdir myapp && cd myapp

npm init -y

npm install express

Create app.js:

javascript

CopyEdit

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Hello from the development environment!'));

app.listen(3000, () => console.log('App running on port 3000'));

Run it:

bash

CopyEdit

node app.js

10. Write and upload your firstsource code file to Github

-> **Step 1: Write a Simple Source Code File**

Let's use a basic Python example.

Create a file hello.py:

python

CopyEdit

hello.py

print("Hello, GitHub!")

Step 2: Create a New GitHub Repository

1. Go to https://github.com and log in.
2. Click **"New"** or **"+" → "New repository"**.
3. Name it something like first-code-upload.
4. Optionally add a description.
5. Keep it **Public** or **Private**.
6. Do **NOT** initialize with README (you'll upload your file manually).
7. Click **Create repository**.

Step 3: Upload Code Using Git (Command Line)

Open Terminal or Command Prompt:

bash

CopyEdit

Clone the empty repo

```
git clone https://github.com/your-username/first-code-upload.git

cd first-code-upload

# Copy your file into this folder
cp /path/to/hello.py .

# Add and commit the file
git add hello.py
git commit -m "Initial commit: Added hello.py"

# Push to GitHub
git push origin main
```

If this is your first time using Git, set your name/email:

```
bash
CopyEdit
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

Step 4: Verify the Upload

1. Go back to your GitHub repository.
2. You should now see hello.py in the file list.
3. Click it to view the code.

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

1. Create a GitHub repository
2. Make code changes locally
3. Commit and push changes to GitHub

Prerequisites

Make sure you have:

- A GitHub account
- Git installed on your system
- A text editor (e.g., VS Code)

Step-by-Step Instructions

---

1. Create a New GitHub Repository

1. Go to https://github.com
2. Click **"+"** in the top-right → Click **"New repository"**
3. Fill in:
    - **Repository name** (e.g., my-first-repo)
    - Description (optional)
    - Set visibility: Public or Private
4. Uncheck **"Initialize this repository with a README"** (optional)
5. Click **Create repository**

---

2. Set Up the Repository Locally

Open your terminal/command prompt:

bash

CopyEdit

```
# Clone the empty repo
git clone https://github.com/your-username/my-first-repo.git

# Navigate into the repo folder
cd my-first-repo

# Create a file
echo "print('Hello GitHub!')" > hello.py
```

---

3. Commit and Push Code Changes

```
bash
CopyEdit
# Check the status of changes
git status

# Stage the new file
git add hello.py

# Commit the changes with a message
git commit -m "Add hello.py with hello message"

# Push to GitHub
git push origin main
```

---

4. View Your Code on GitHub

1. Go back to your repository page on GitHub.

2. Refresh the page.
3. You will see your hello.py file with your commit message.

---

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

-> Here's a step-by-step guide to:

1. Create a GitHub Student Developer Pack account
2. Start and collaborate on a small project with a classmate

Step 1: Create a GitHub Student Account

1. Sign up for GitHub

- Go to https://github.com/join
- Create a username, enter your email, and set a password.
- Choose free plan to start.

---

2. Apply for GitHub Student Developer Pack

1. Go to: https://education.github.com/pack
2. Click "Get Student Benefits"
3. Fill out the form:
    - Use your school/university email address (e.g., you@college.edu)
    - Upload proof (e.g., student ID card, admission letter)
    - Enter school name and expected graduation date

Approval usually takes 1–5 days

Step 2: Start a Collaborative Project with a Classmate

1. Create a New Repository

    1. Go to https://github.com
    2. Click “+” ➜ “New repository”
    3. Name it something like student-collab-project
    4. Choose:
        - Add a README.md
        - Add a .gitignore (e.g., Python, Node, etc.)
    5. Click Create Repository

---

2. Add Your Classmate as a Collaborator

    1. Go to the Settings tab of your repo
    2. Click Collaborators and teams
    3. Click “Add people”
    4. Enter your classmate's GitHub username
    5. Click Invite

They will get an email to accept and join your repo.

---

3. Clone the Repository and Start Coding

On both of your computers:

bash

CopyEdit

git clone https://github.com/your-username/student-collab-project.git

cd student-collab-project

Add a file like main.py or index.html and start collaborating!

---

4. Collaborate Using Git

Example workflow:

bash

CopyEdit

```
# Add or update a file
echo "print('Hello from me!')" > hello.py

# Stage the file
git add hello.py

# Commit changes
git commit -m "Added hello.py"

# Push to GitHub
git push origin main
```

---

5. Pull Changes Made by Your Classmate

bash

CopyEdit

```
git pull origin main
```

---

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

-> Here's a list of software types and examples, categorized as system, application, or utility software. This list is based on common software usage and may vary depending on individual needs.

System Software:

- Operating System:

Windows (or macOS/Linux) - This is the foundational software that manages all other programs and hardware resources.

- Device Drivers:

(e.g., printer drivers, graphics card drivers) - These enable communication between the operating system and hardware components.

- Middleware:

(e.g., database connectors, APIs) - Software that connects different applications or systems.

Application Software:

- Web Browsers: Chrome, Firefox, Safari - Used for accessing and navigating the internet.
- Word Processors: Microsoft Word, Google Docs - Used for creating and editing text documents.
- Spreadsheet Software: Microsoft Excel, Google Sheets - Used for organizing and analyzing data in a table format.
- Email Clients: Outlook, Gmail - Used for sending and receiving emails.
- Media Players: VLC, Windows Media Player - Used for playing audio and video files.
- Graphics Software: Adobe Photoshop, GIMP - Used for editing and creating images.
- Communication Software: Slack, Skype, Zoom - Used for online communication and collaboration.
- Project Management Software: Asana, Trello - Used for managing tasks and projects.
- Database Software: MySQL, PostgreSQL - Used for managing and organizing large amounts of data.
- Presentation Software: PowerPoint, Google Slides - Used for creating and delivering presentations.

Utility Software:

- Antivirus Software: Protects the computer from malicious software.
- File Management Tools: Allows for organizing, copying, and managing files.
- Compression Tools: Used to reduce the size of files for storage or transmission.
- Disk Management Tools: Helps manage hard drives and partitions.
- Disk Cleanup Tools: Removes temporary files and unnecessary data from the hard drive.
- Disk Defragmenters: Optimizes the organization of files on a hard drive.

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

-> Here's a hands-on Git tutorial to practice cloning, branching, and merging a repository. You'll learn how to:

Clone a GitHub repo
Create and switch branches
Make changes and commit
Merge branches

Prerequisites

- Git installed
- GitHub account
- A terminal or command prompt

Step-by-Step Git Practice

Step 1: Create a New GitHub Repository

1. Go to https://github.com
2. Click "+" ➜ "New repository"

3. Name it: git-practice
4. Check “Add a README file”
5. Click Create repository

Step 2: Clone the Repository to Your Local Machine

bash

CopyEdit

Clone your repository

git clone https://github.com/your-username/git-practice.git

Move into the cloned folder

cd git-practice

Step 3: Create and Switch to a New Branch

bash

CopyEdit

Create a new branch

git branch feature-hello

Switch to the new branch

git checkout feature-hello

You can also do this in one step:

bash

CopyEdit

git checkout -b feature-hello

Step 4: Make Changes and Commit

Let’s create a new file:

bash

CopyEdit

```
echo "print('Hello from a new branch!')" > hello.py
```

Now commit the changes:

bash

CopyEdit

```
git add hello.py
git commit -m "Add hello.py file"
```

Step 5: Merge the Branch Back to main

1. Switch back to main branch:

bash

CopyEdit

```
git checkout main
```

2. Merge the feature branch:

bash

CopyEdit

```
git merge feature-hello
```

Step 6: Push Changes to GitHub

bash

CopyEdit

```
git push origin main
```

You’ll now see your changes reflected in your GitHub repo.

---

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

-> **1. Introduction**

Application software refers to programs designed to perform specific tasks for users. Unlike system software that manages hardware, application software helps users achieve personal or professional goals, from writing documents to analyzing data or managing communication.

---

2. Types of Application Software

1. Word Processing Software

- **Examples**: Microsoft Word, Google Docs
- **Purpose**: Create, edit, and format text documents
- **Productivity Impact**:
    - Enables faster typing and editing
    - Built-in spell check and formatting tools
    - Cloud collaboration with others in real-time

---

2. Spreadsheet Software

- **Examples**: Microsoft Excel, Google Sheets
- **Purpose**: Perform calculations, manage data, create charts
- **Productivity Impact**:
    - Automates calculations using formulas
    - Organizes large datasets
    - Visualizes data through graphs and charts

---

3. Presentation Software

- **Examples**: Microsoft PowerPoint, Google Slides
- **Purpose**: Create visual slide-based presentations
- **Productivity Impact**:
    - Communicates ideas clearly and visually
    - Enhances team collaboration in meetings
    - Speeds up the creation of professional content

---

4. Database Management Software

- **Examples**: Microsoft Access, MySQL, Oracle
- **Purpose**: Store, retrieve, and manage data efficiently
- **Productivity Impact**:
    - Centralized data management
    - Ensures data integrity and security
    - Supports data-driven decision-making

---

5. Communication Software

- **Examples**: Microsoft Teams, Zoom, Slack, WhatsApp
- **Purpose**: Enable messaging, video calls, and collaboration
- **Productivity Impact**:
    - Enhances remote teamwork
    - Reduces need for in-person meetings
    - Provides instant updates and feedback

---

6. Multimedia Software

- **Examples**: VLC Media Player, Adobe Photoshop, Canva
- **Purpose**: View, create, or edit multimedia files
- **Productivity Impact**:
    - Supports content creation (images, videos, audio)
    - Improves digital communication and marketing
    - Enables creative and technical expression

---

7. Project Management Software

- **Examples**: Trello, Asana, Microsoft Project
- **Purpose**: Plan, schedule, and manage tasks and resources
- **Productivity Impact**:
    - Keeps teams organized and on track
    - Tracks deadlines and progress
    - Improves project efficiency and transparency

---

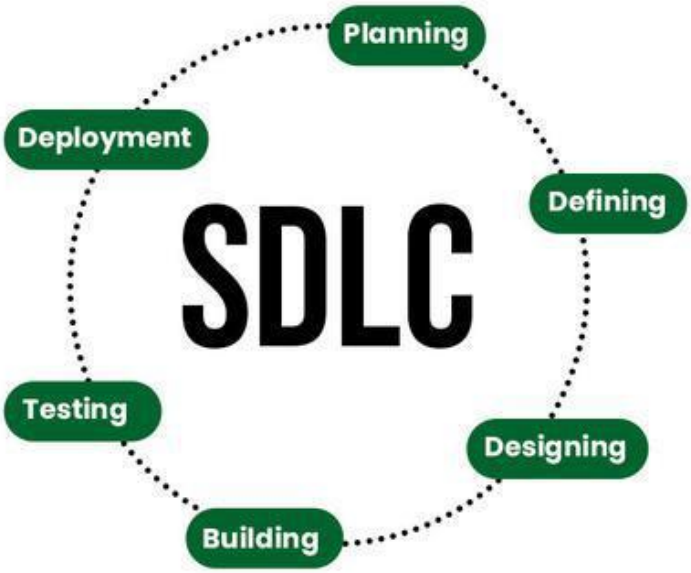
8. Web Browsers

- **Examples**: Google Chrome, Mozilla Firefox
- **Purpose**: Access and navigate the internet
- **Productivity Impact**:
    - Enables quick access to information and tools
    - Supports research, online learning, and remote tools
    - Allows extension-based customization for workflow

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

-> SDLC is a process followed for software building within a software organization**.** SDLC consists of a precise plan that describes how to develop,

maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.



1. Requirement Analysis

- Gather user needs.
- Define system expectations.

2. System Design

- Define system architecture.
- Create design specifications (UI, database, etc.).

3. Implementation (Coding)

- Developers write code according to design specs.

4. Testing

- Detect bugs and verify software quality.
- Includes unit testing, integration testing, etc.

5. Deployment

- Deliver the software to users or release to production.

6. Maintenance

- Fix issues, apply updates, and enhance features over time.

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

-> 1.1 Purpose

This document outlines the functional and non-functional requirements for a Library Management System (LMS) that will automate the activities of a library such as book issue/return, user management, and inventory tracking.

1.2 Scope

The system is designed for use by librarians, library staff, and registered users. It includes features like:

- Book registration and cataloging
- Member registration
- Book issuance and return
- Fine calculation for late returns
- Search and reporting functionalities

1.3 Intended Audience

- Developers
- Project Managers
- QA Engineers
- Library Staff and IT Administrators

---

2. Functional Requirements

2.1 User Authentication

- Admin and staff must log in with valid credentials.
- Role-based access (Admin, Staff, Member).

2.2 Book Management

- Add, edit, delete book records.
- Search books by title, author, or ISBN.
- View available and borrowed copies.

2.3 Member Management

- Register new members with personal details.
- View/update member profile.
- Track member borrowing history.

2.4 Book Issuance & Return

- Issue books to members.
- Return books and update stock.
- Calculate fines based on late return.

2.5 Search Functionality

- Search by title, author, genre, or member ID.
- Filter results (available/issued).

2.6 Reports

- Daily/weekly/monthly issuance reports.
- Overdue books and fine summary.
- Most issued books.

---

3. Non-Functional Requirements

3.1 Performance

- Should support up to 100 concurrent users.
- System response time < 2 seconds for search queries.

3.2 Security

- Passwords should be encrypted.
- Only authorized users can access or modify data.

3.3 Usability

- Clean, intuitive interface.
- Minimal training required for users.

3.4 Maintainability

- System should be modular and easy to update or enhance.

3.5 Availability

- The system should be available 99% of the time (excluding maintenance).

---

4. Assumptions and Constraints

- The system will be web-based.
- Users must have internet access and a modern browser.
- Hosted on a local server or cloud environment.

---

5. System Requirements

5.1 Hardware

- Minimum 4 GB RAM, Dual-core CPU (client)
- Minimum 8 GB RAM, Quad-core CPU (server)

5.2 Software

- OS: Windows/Linux
- Backend: MySQL/PostgreSQL

- Frontend: HTML/CSS/JavaScript
- Server: Node.js / Django / PHP

18. Perform a functional analysis for an online shopping system

-> Here's a functional analysis of an Online Shopping System that outlines the major functions the system must perform to meet user and business needs.

---

1. Functional Requirements

Functional requirements describe what the system should do — the core functionalities.

A. User Management

- User Registration: New users can sign up by providing personal and contact details.
- User Login/Logout: Existing users can securely log in and out.
- Profile Management: Users can update their personal info, change passwords, and view order history.

B. Product Catalog Management

- Product Listing: Display categorized list of products with name, price, image, and description.
- Search and Filter: Allow users to search products by keyword and filter by category, brand, price range, etc.
- Product Details: Show detailed information about each product on a separate page.

C. Shopping Cart

- Add to Cart: Users can add products to a virtual shopping cart.
- Update Cart: Modify quantities or remove items from the cart.
- View Cart: Review items, prices, and total cost before checkout.

D. Checkout Process

- Shipping Details: Collect address and delivery preferences.
- Payment Integration: Support for multiple payment options like credit card, debit card, UPI, net banking, etc.
- Order Confirmation: Generate and show an order summary with invoice.

E. Order Management

- Order Tracking: Users can track their order status (e.g., confirmed, packed, shipped, delivered).
- Order History: View list of past orders with details.
- Cancellation/Return: Request order cancellation or initiate product return.

F. Admin Functions

- Product Management: Admins can add, update, or delete product listings.
- Order Management: Admins can update order statuses and handle returns/refunds.
- User Management: Admins can view user accounts and manage user-related issues.
- Reports and Analytics: Generate reports on sales, inventory, and user behavior.

System Functional Flow (Simplified)

1. User registers/logs in.
2. Browses or searches for products.
3. Adds items to cart.
4. Proceeds to checkout → fills shipping and payment info.
5. Order is placed and confirmation is shown.
6. User can track the order.
7. Admin monitors order processing and fulfillment.

Supporting Functions

- Email/SMS Notifications (e.g., order confirmation, delivery updates)
- Wishlist Management (users can save items for later)
- Discounts & Coupons (apply promo codes during checkout)
- Review & Ratings (users can review products post-purchase)
- Inventory Management (auto-update product availability)

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

-> Designing a basic system architecture for a Food Delivery App involves identifying the major components and how they interact. Here's a layered, modular architecture suitable for scalability and performance.

Basic System Architecture for a Food Delivery App

---

1. High-Level Components

The system involves four primary actors:

1. Customer
2. Restaurant
3. Delivery Agent
4. Admin

---

2. Architectural Layers

A. Frontend Layer (Client Applications)

- Customer App/Web: Browse restaurants, order food, track delivery.
- Restaurant Dashboard: Accept/prepare orders.
- Delivery Agent App: Accept delivery tasks, GPS tracking.

- Admin Panel: Manage restaurants, users, and logistics.

B. Backend Layer (Application Services)

Handles the business logic and communication between the database and frontend.

Key microservices/modules:

Module	Functions
Authentication Service	Login, Signup, JWT Token
User Service	Profile, Address, Preferences
Restaurant Service	Menu, Availability, Ratings
Order Service	Cart, Order Placement, Tracking
Delivery Service	Assign riders, ETA calculations
Payment Service	Payment gateway integration, Wallet
Notification Service	SMS/Email/Push alerts
Review & Rating Service	Collect & display feedback

C. Database Layer (Data Storage)

- Relational DB (e.g., PostgreSQL, MySQL)
  For structured data: Users, Orders, Payments
- NoSQL DB (e.g., MongoDB)
  For unstructured data: Menus, Reviews, Images
- Caching (Redis/Memcached)
  For fast-access data like nearby restaurants or order status
- Blob Storage (e.g., AWS S3)
  For storing media (images, menus)

D. External Services / APIs

- Maps API (e.g., Google Maps): For route & delivery tracking
- Payment Gateways (e.g., Razorpay, Stripe): For secure transactions
- SMS/Email API (e.g., Twilio, SendGrid): For notifications
- Cloud Hosting (AWS/Azure/GCP): For deployment & scalability

---

3. Data Flow Summary

1. Customer logs in, browses menu, places an order.
2. Order Service validates and stores the order.
3. Restaurant is notified via the dashboard/app.
4. Delivery Service assigns an agent using location tracking.
5. Payment Service processes payment securely.
6. Notification Service alerts customer and agent.
7. Delivery Agent picks up the food and updates order status.
8. Customer receives food and provides feedback.

---

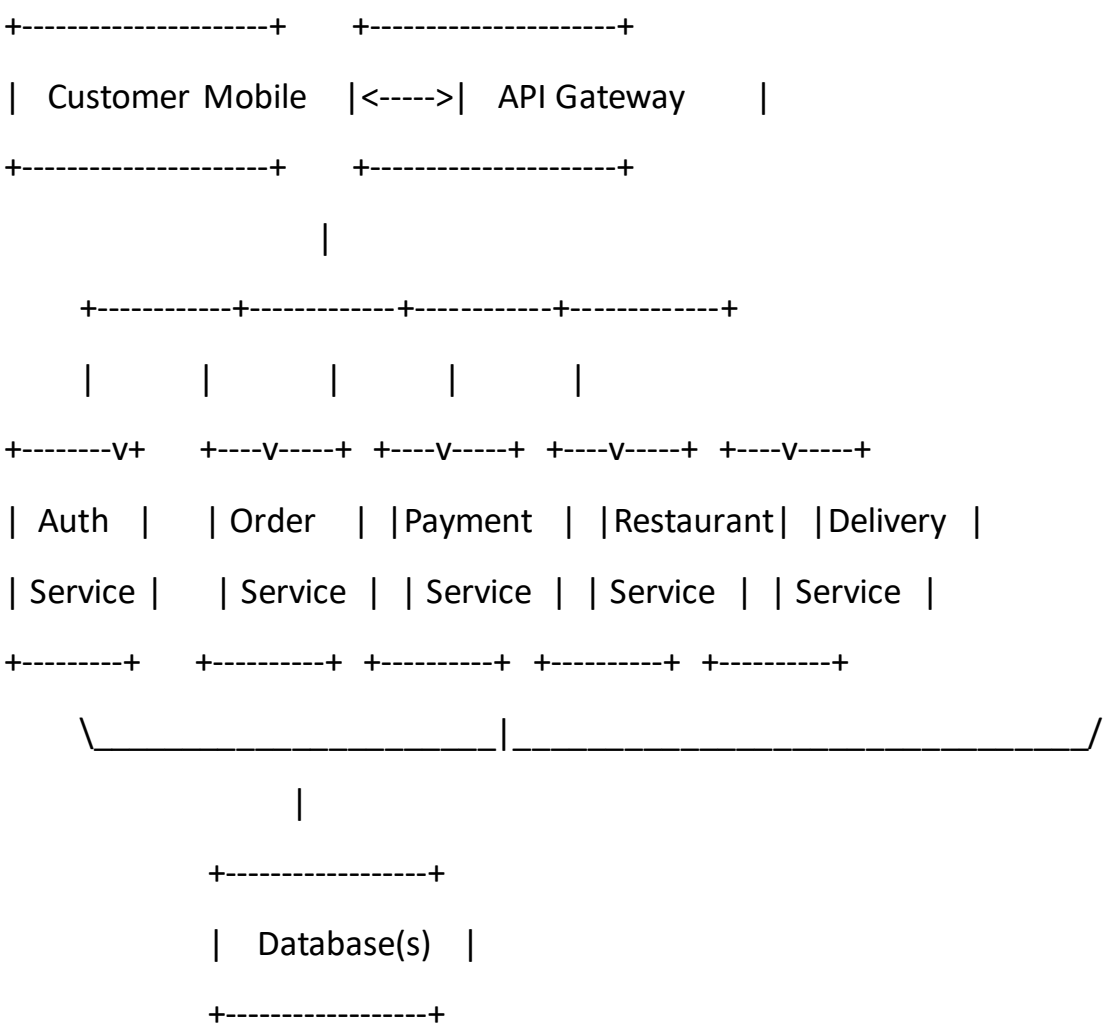
4. Security & Performance

- HTTPS & Data Encryption
- Rate Limiting & API Gateway
- Load Balancers
- Role-Based Access Control (RBAC)
- Scalable Microservices
- Database Backup & Failover Systems

Optional Diagram (Text View)

plaintext

CopyEdit



20. Develop test cases for a simple calculator program

-> Here's a list of **test cases** for a **simple calculator program** that supports basic arithmetic operations such as **addition, subtraction, multiplication, and division**. These test cases can be used for **manual testing** or as a basis for **automated unit tests**.

✅ **Test Case Table**

Test Case ID	Description	Input	Expected Output	Remarks
TC_001	Addition of two positive integers	5 + 3	8	Basic addition test
TC_002	Addition of negative and positive	-5 + 3	-2	Negative number support
TC_003	Subtraction of two numbers	10 - 4	6	Basic subtraction test
TC_004	Subtraction resulting in negative	4 - 10	-6	Result should be negative
TC_005	Multiplication of two numbers	7 * 6	42	Basic multiplication
TC_006	Multiplication with zero	0 * 9	0	Zero multiplication test
TC_007	Division of two numbers	20 / 4	5	Basic division test
TC_008	Division by 0	9 / 0	Error / Exception	Must handle divide-by-zero
TC_009	Floating point division	5 / 2	2.5	Decimal division test
TC_010	Addition of two floating numbers	1.2 + 3.4	4.6	Precision check
TC_011	Invalid characters in input	3 + a	Error	Input validation test
TC_012	No input	""	Error	Input should not be empty
TC_013	Complex expression (if supported)	2 + 3 * 4	14 or Error	Depending on parser logic
TC_014	Whitespace handling	4 + 5	9	Should handle spaces

Test Case ID	Description	Input	Expected Output	Remarks
TC_015	Negative multiplication	-3 * -2	6	Negative times negative
TC_016	Chained operations (if supported)	5 + 2 - 1	6	Evaluate left to right

---

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

-> A critical software maintenance situation arose with a large e-commerce platform after a major security vulnerability was discovered in its payment processing system. This required immediate action to patch the flaw and prevent potential data breaches and financial losses for both the company and its customers.

Case Details:

- **Vulnerability:**

A critical security flaw was found in the e-commerce platform's payment gateway integration, specifically within the encryption protocols used to handle sensitive customer credit card information. This flaw could potentially allow malicious actors to intercept and steal credit card details during online transactions.

- **Impact:**

The vulnerability posed a significant risk of data breaches, financial fraud, and reputational damage for the e-commerce company. A successful exploit could lead to substantial financial losses due to fraudulent transactions and legal liabilities.

- **Maintenance Response:**

The development team immediately initiated a corrective maintenance process, prioritizing the following actions:

- **Patching the Vulnerability:** A security patch was developed and rigorously tested to address the identified flaw in the payment processing system. This patch involved updating the encryption protocols to a more secure standard and implementing additional security measures.
- **Emergency Deployment:** The security patch was deployed to the production environment with minimal downtime to mitigate further risk. The platform was temporarily taken offline during the deployment to prevent any potential issues during the patching process.
- **Communication and Transparency:** The company proactively communicated with its customers, informing them of the security issue and the measures taken to resolve it. They also provided guidance on how customers could further protect their accounts.

- **Outcomes:**
  - **Data Protection:** The security patch effectively addressed the identified vulnerability, preventing further unauthorized access to customer data.
  - **Restored Trust:** The prompt and transparent response helped restore customer confidence and mitigated potential reputational damage.
  - **Prevented Financial Losses:** The rapid patching of the vulnerability prevented potential financial losses from fraudulent transactions and legal liabilities.
  - **Lessons Learned:** The incident highlighted the importance of robust security practices, regular security audits, and a well-defined incident response plan.

22. DFD (Data Flow Diagram)

-> A **DFD** is a graphical representation of the flow of data in a system. It shows:

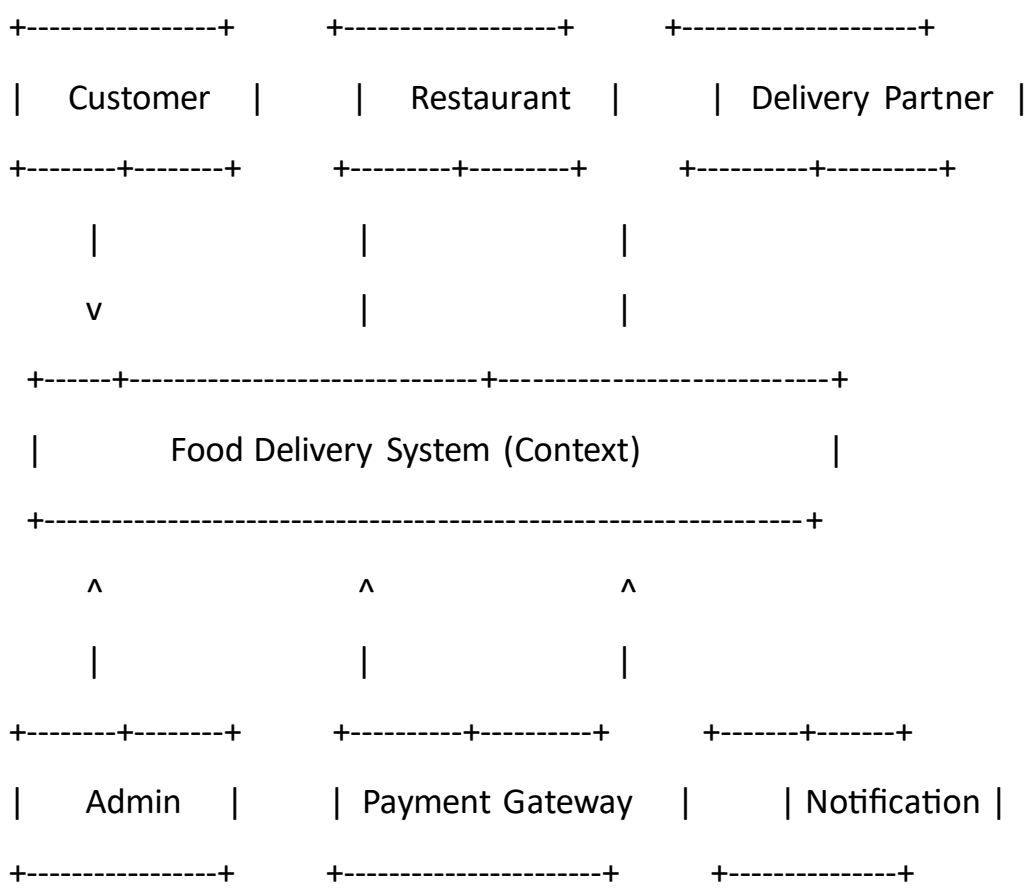
- **Processes** (circles or rounded rectangles)

- **Data Stores** (open-ended rectangles)
- **External Entities** (squares)
- **Data Flows** (arrows)

Level 0 DFD (Context Diagram) – Food Delivery App

pgsql

CopyEdit



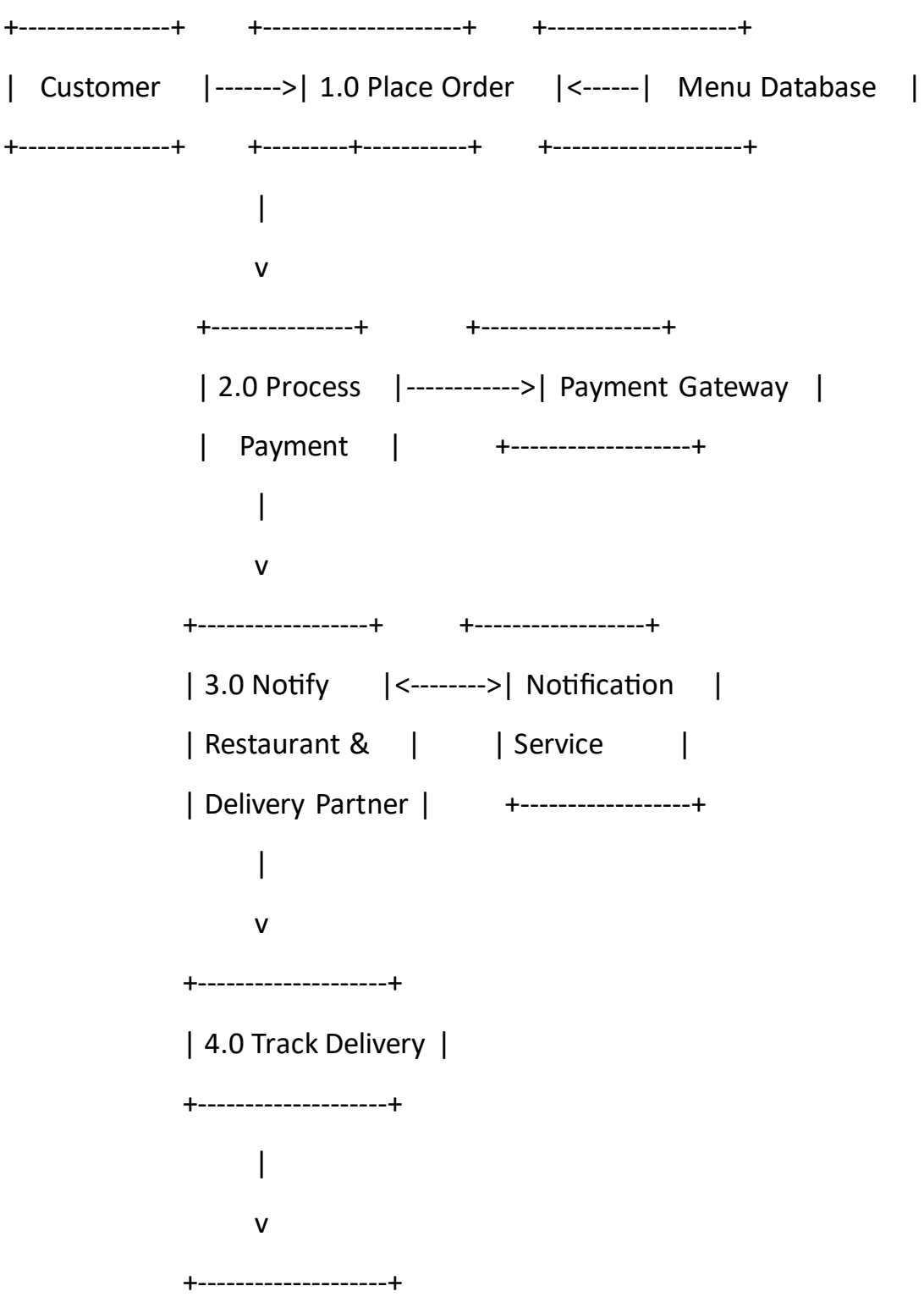
Explanation:

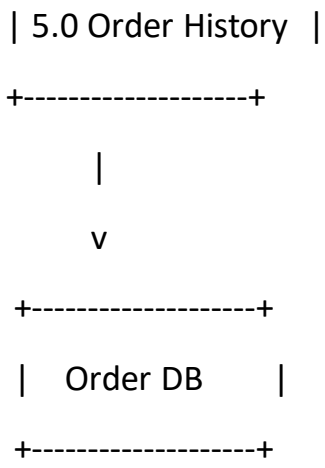
- **Customer** sends orders → system
- **System** sends order → Restaurant and assigns delivery to partner
- **Payment Gateway** handles transaction data
- **Admin** manages the entire system
- **Notifications** sent to customer and delivery partner

Level 1 DFD – Food Delivery App

pgsql

CopyEdit





Data Stores Used:

- **Menu Database**: Contains available items, prices
- **Order DB**: Stores placed orders
- **Notification Service**: Push/SMS/Email handling

Tools to Draw a DFD:

You can use:

- draw.io
- Lucidchart
- Microsoft Visio
- Creately

---

23. Create a DFD for a hospital management system.

-> Here is a detailed **Data Flow Diagram (DFD)** for a **Hospital Management System**, including:

- **Level 0 (Context Diagram)**: Overall system view
- **Level 1 DFD**: Major sub-processes and data flow

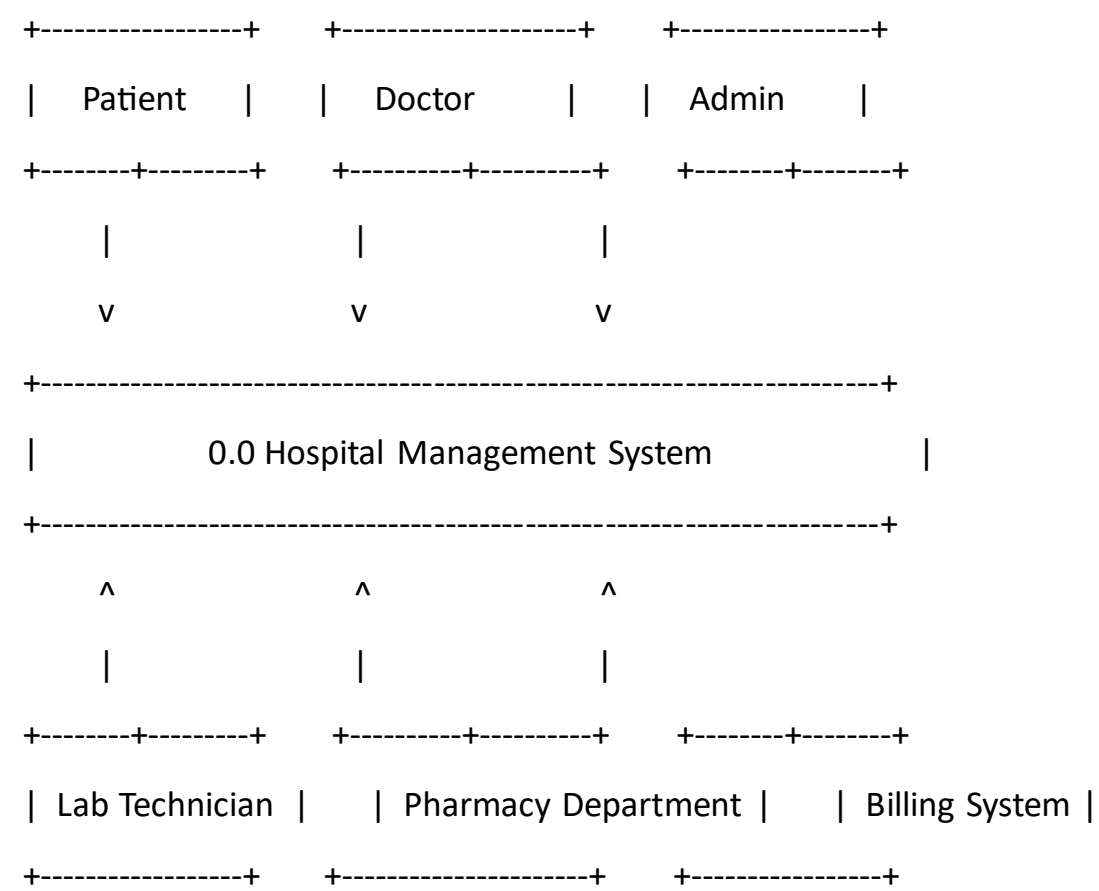
DFD – Hospital Management System

Level 0 DFD (Context Diagram)

This diagram shows the hospital system as a **single process** with all external entities:

pgsql

CopyEdit

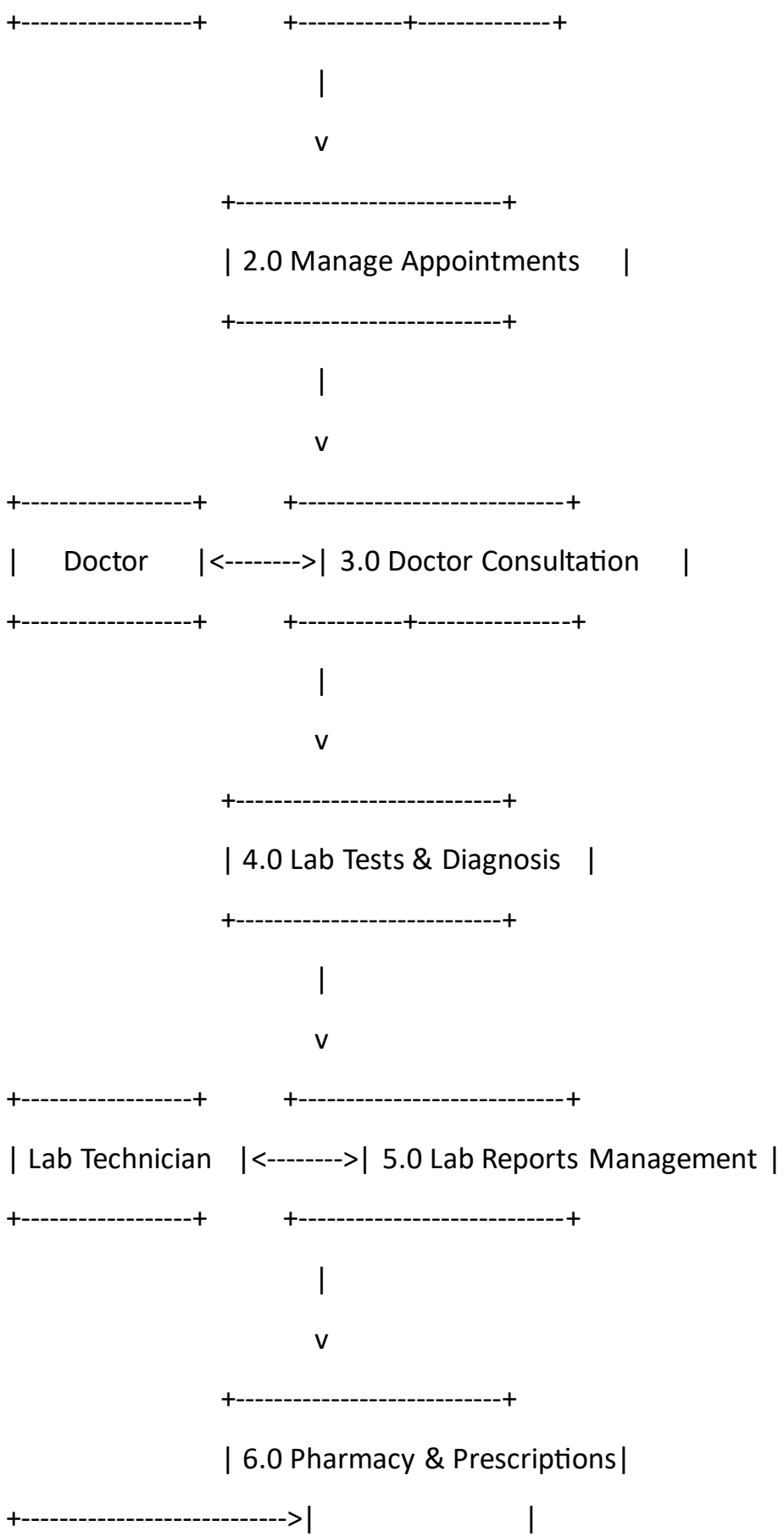


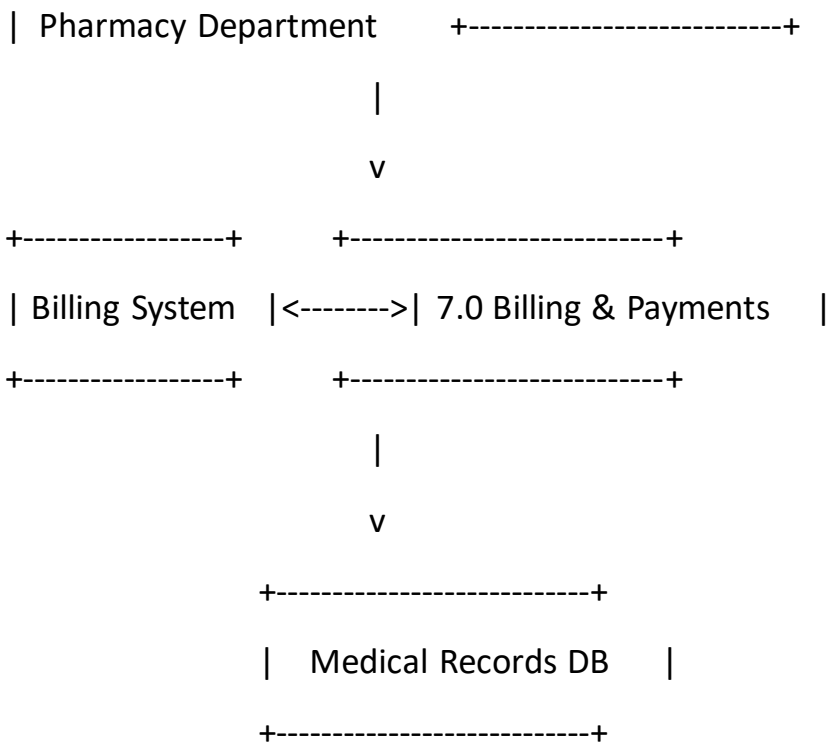
Level 1 DFD – Hospital Management System

Breaks down the system into **core modules**:

sql

CopyEdit


2.0 Manage Appointments
Doctor
3.0 Doctor Consultation
4.0 Lab Tests & Diagnosis
Lab Technician
5.0 Lab Reports Management
6.0 Pharmacy & Prescriptions



Data Stores Used in Level 1:

- **Patient Records DB**
- **Appointment Schedule DB**
- **Prescription DB**
- **Lab Test Results DB**
- **Billing DB**
- **User Authentication DB**

Key Processes Explained:

Process No.	Name	Description
1.0	Register Patient	Stores patient info and medical history
2.0	Manage Appointments	Schedule/edit/cancel appointments
3.0	Doctor Consultation	Access patient records, enter diagnoses

Process No.	Name	Description
4.0	Lab Tests & Diagnosis	Request & process test results
5.0	Lab Reports Management	Store and deliver reports
6.0	Pharmacy & Prescriptions	Issue and manage medicine orders
7.0	Billing & Payments	Calculate charges and generate invoices

---

Tools to Draw This DFD:

You can create a clean diagram using:

- draw.io
- Lucidchart
- Microsoft Visio
- Creately

24. : Build a simple desktop calculator application using a GUI library.

-> Simple Desktop Calculator Using Tkinter (Python)

Features:

- GUI interface
- Basic operations: +, -, *, /
- Clear and equals buttons

Python Code:

python

CopyEdit

```
import tkinter as tk


# Main application window
```

```
root = tk.Tk()
root.title("Simple Calculator")
root.geometry("300x400")
root.resizable(False, False)

# Entry widget to display expressions
entry = tk.Entry(root, width=20, font=("Arial", 24), bd=10, insertwidth=2,
borderwidth=4, relief="ridge", justify="right")
entry.grid(row=0, column=0, columnspan=4)

# Function to handle button click
def click(value):
    current = entry.get()
    entry.delete(0, tk.END)
    entry.insert(0, current + value)

# Function to clear the entry
def clear():
    entry.delete(0, tk.END)

# Function to evaluate the expression
def evaluate():
    try:
        result = str(eval(entry.get()))
        entry.delete(0, tk.END)
        entry.insert(0, result)
    except:
```

```
        entry.delete(0, tk.END)
        entry.insert(0, "Error")

# Button layout
buttons = [
    ('7',1,0), ('8',1,1), ('9',1,2), ('/',1,3),
    ('4',2,0), ('5',2,1), ('6',2,2), ('*',2,3),
    ('1',3,0), ('2',3,1), ('3',3,2), ('-',3,3),
    ('0',4,0), ('.',4,1), ('=',4,2), ('+',4,3),
    ('C',5,0)
]

# Create buttons dynamically
for (text, row, col) in buttons:
    if text == "=":
        tk.Button(root, text=text, padx=20, pady=20, font=("Arial", 18),
command=evaluate).grid(row=row, column=col, sticky="nsew")
    elif text == "C":
        tk.Button(root, text=text, padx=20, pady=20, font=("Arial", 18),
command=clear).grid(row=row, column=col, columnspan=4, sticky="nsew")
    else:
        tk.Button(root, text=text, padx=20, pady=20, font=("Arial", 18),
command=lambda val=text: click(val)).grid(row=row, column=col,
sticky="nsew")

# Start the GUI loop
root.mainloop()
```

How to Run It:

1. Save the code in a file named calculator.py.
2. Run it using:

nginx

CopyEdit

python calculator.py

Requirements:

- Python 3.x (Tkinter comes pre-installed)

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

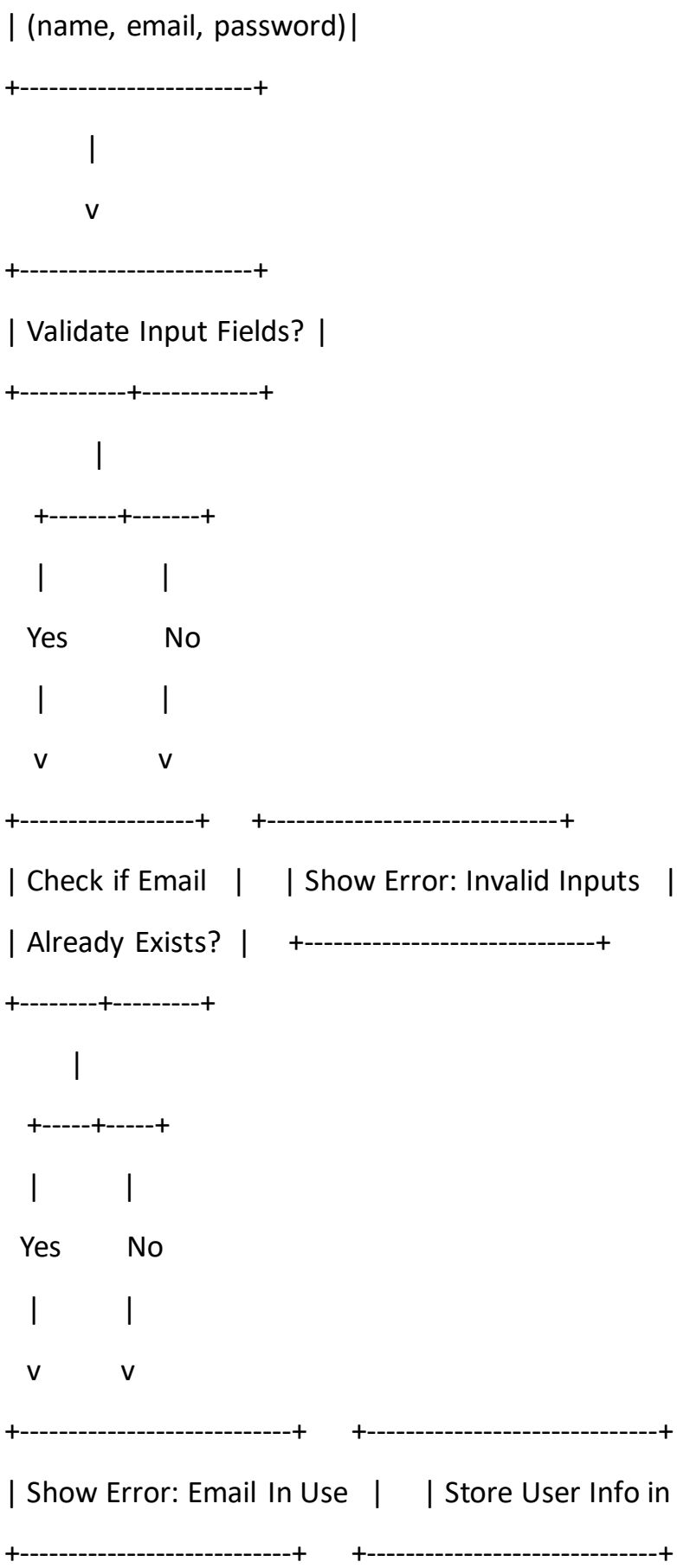
-> Here's a flowchart that represents the logic of a basic online registration system. It outlines the steps from the user's input to the registration confirmation.

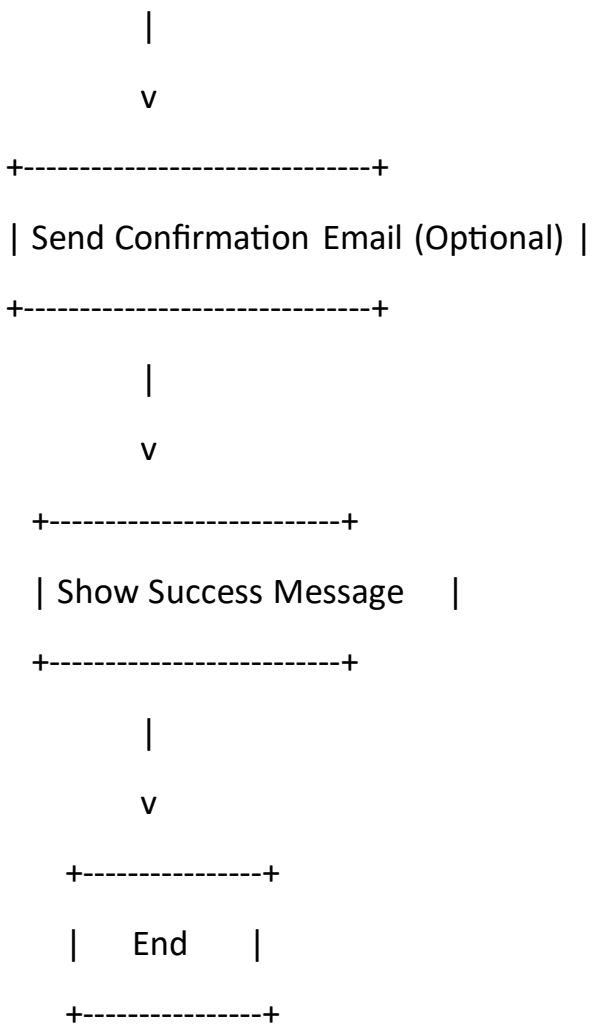
Flowchart: Online Registration System

pgsql

CopyEdit

```
+-----------------------+
| Start Registration    |
+-----------------------+
          |
          v
+-----------------------+
| Enter User Details    |
```


| (name, email, password)|
Validate Input Fields?
Yes
No
Check if Email Already Exists?
Yes
No
Show Error: Email In Use
Store User Info in Database
Show Error: Invalid Inputs



🧩 Main Components Explained:

- Input Validation: Checks if all required fields are filled and properly formatted.
- Duplicate Check: Ensures email is not already registered.
- Data Storage: If valid and unique, stores the user data.
- Feedback: Gives the user an appropriate success or error message.