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

* **c**

**#include <stdio.h>**

**int main() {**

**printf("Hello World");**

**return 0;**

**}**

* **C++**

**#include <iostream>**

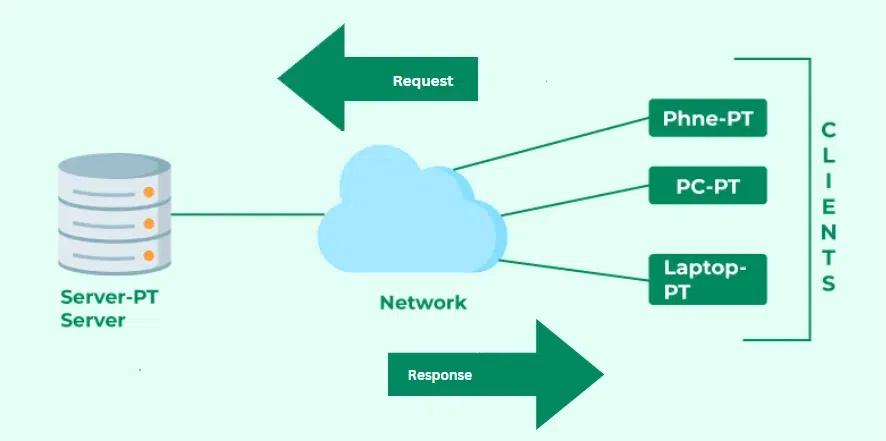
**int main() {**

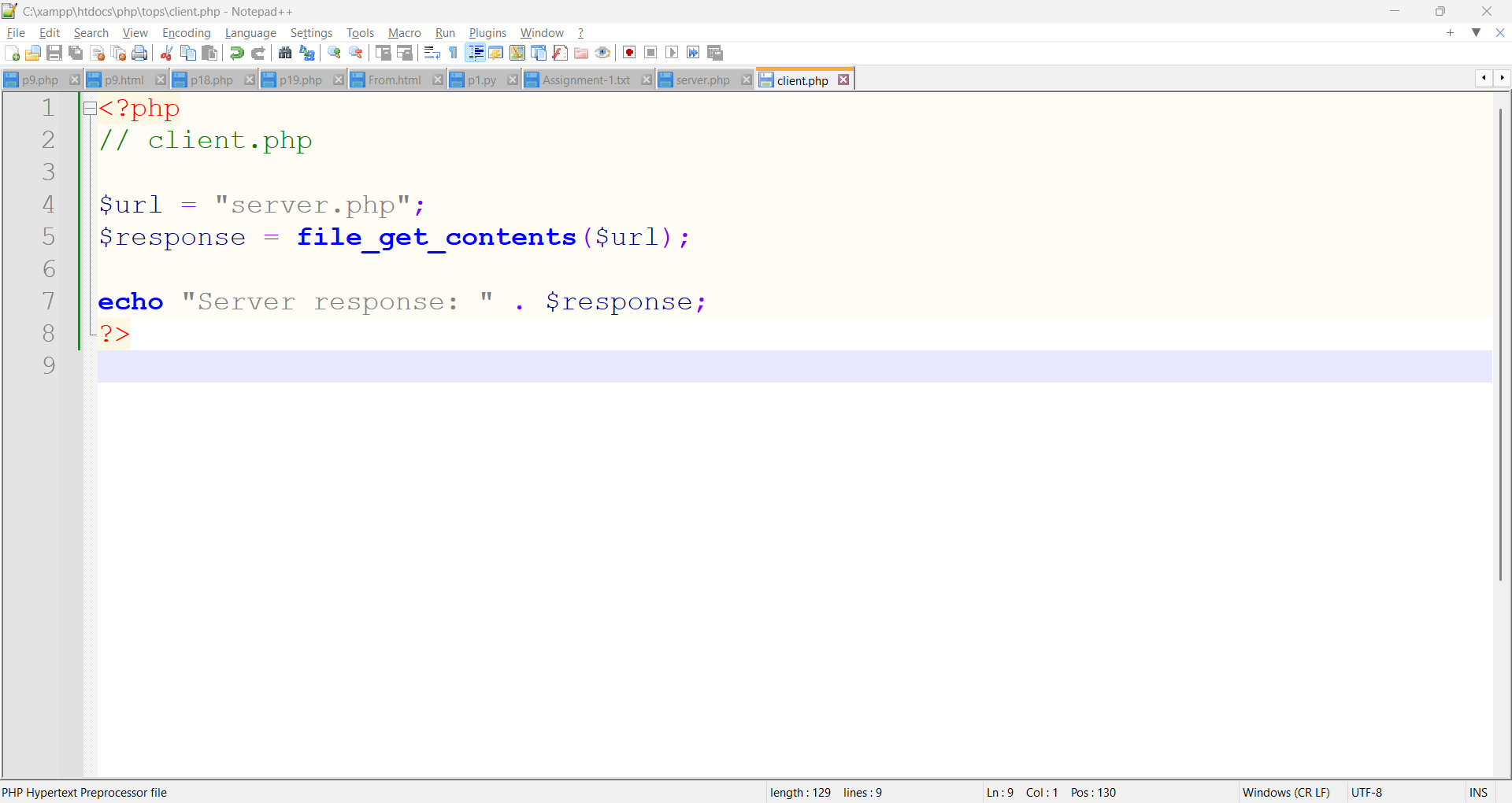
**std::cout << "Hello World";**

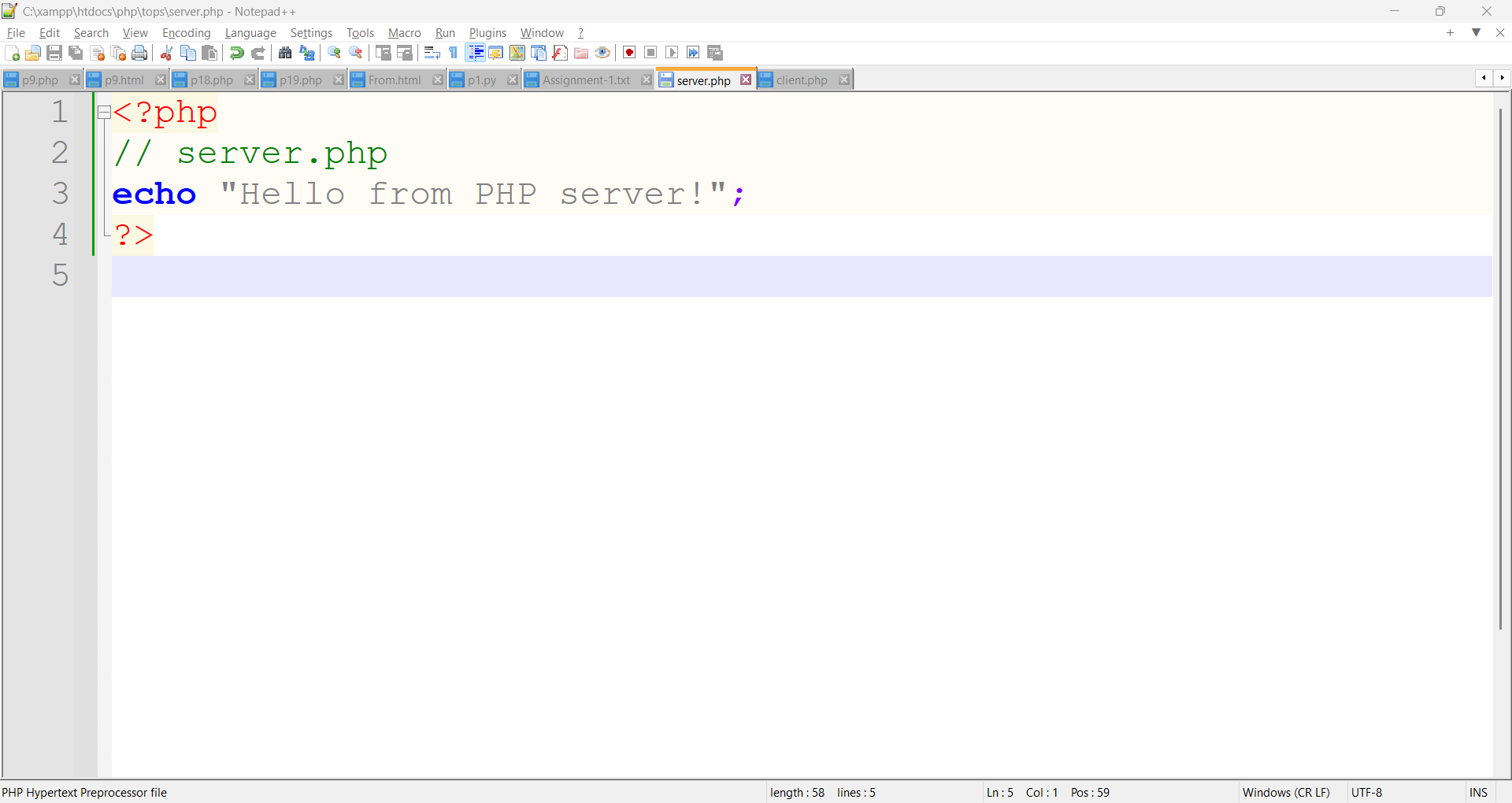
**return 0;**

**}**

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

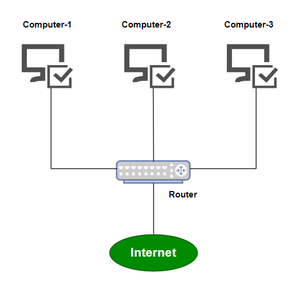


2) Design a simple HTTP client-server communication in any language.



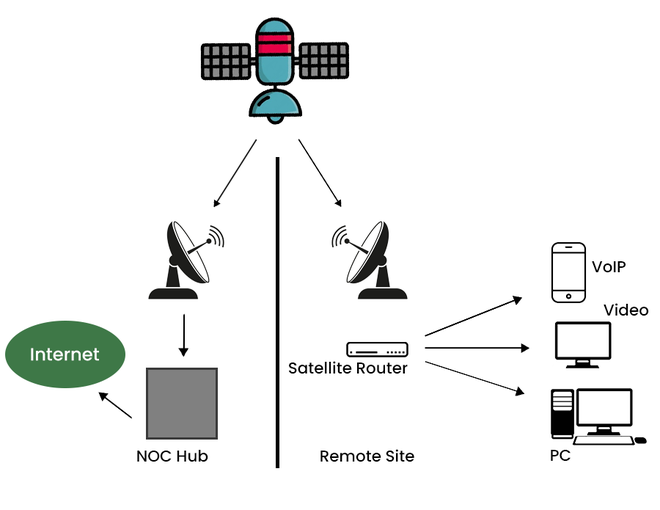
# 3) Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.

* **Broadband Connection**



* **Pros:**
* **Always on (no need to dial in)**
* **Faster than dial-up**
* **Widely available**
* **Cons:**
* **Speed and quality can vary**
* **May slow down during peak hours**

### **Satellite Connection**



* **Pros:**
* **Available in rural or remote areas**
* **No cables needed**
* **Cons:**
* **High latency (slow response time)**
* **Affected by weather**
* **Data limits and lower speed**

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

| Task | Command |
| --- | --- |
| HTTP request | curl http://example.com |
| HTTPS request | curl https://example.com |
| FTP download | curl ftp://... -o filename.zip |

# 5) Identify and explain three common application security vulnerabilities. Suggest possible solutions.

* **1. SQL Injection**

**Explanation:**  
SQL injection occurs when an attacker inserts malicious SQL code into a query via input fields (e.g., login forms). This can allow unauthorized access to a database or manipulation of data.

* **Solutions**
* Validate and sanitize all user inputs.
* Employ web application firewalls (WAFs).
* **2**. **Cross-Site Scripting (XSS)**

**xplanation:**  
XSS allows attackers to inject malicious scripts into web pages viewed by other users. It can be used to steal session cookies, deface websites, or redirect users.

* **Solutions:**
* Use Content Security Policy (CSP).
* Validate and sanitize all input fields.
* **3.** **Broken Authentication**

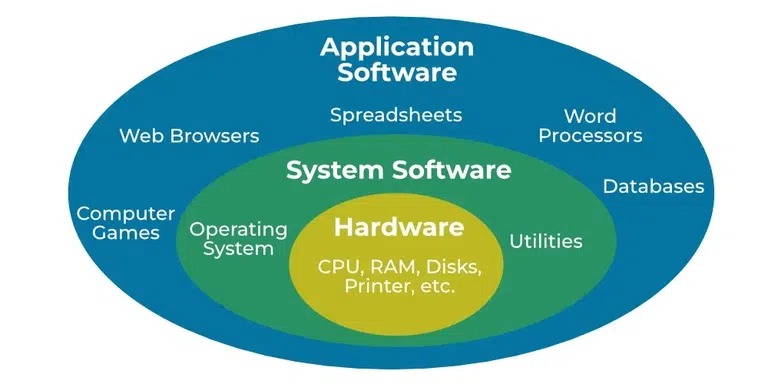
Occurs when authentication mechanisms are improperly implemented, allowing attackers to compromise passwords, keys, or session tokens.

* **Solutions:**
* Implement multi-factor authentication (MFA).
* Enforce strong password policies.
* Use secure session management (e.g., rotating session IDs after login).

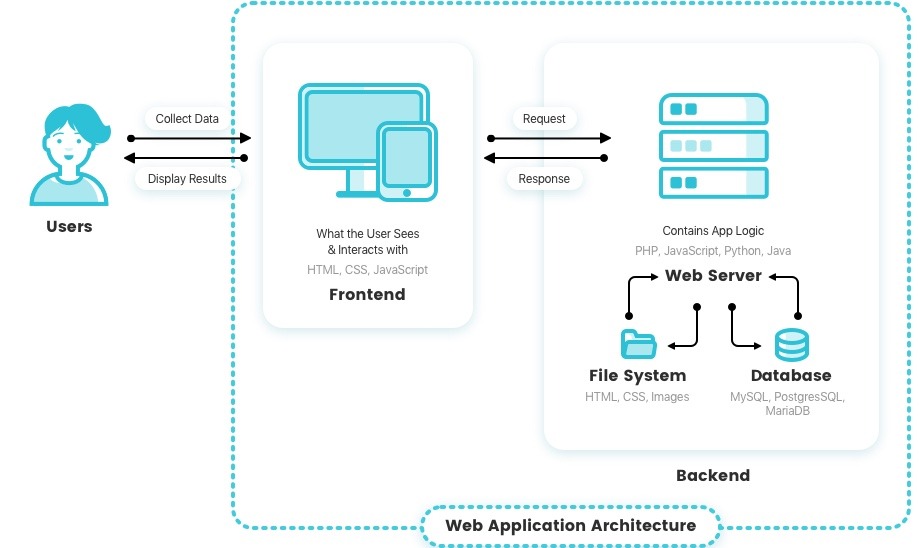
# 6)LAB EXERCISE: Identify and classify 5 applications you use daily as either system software or application software.

* **Task:** Identify and classify 5 applications used daily as **system software** or **application software**.

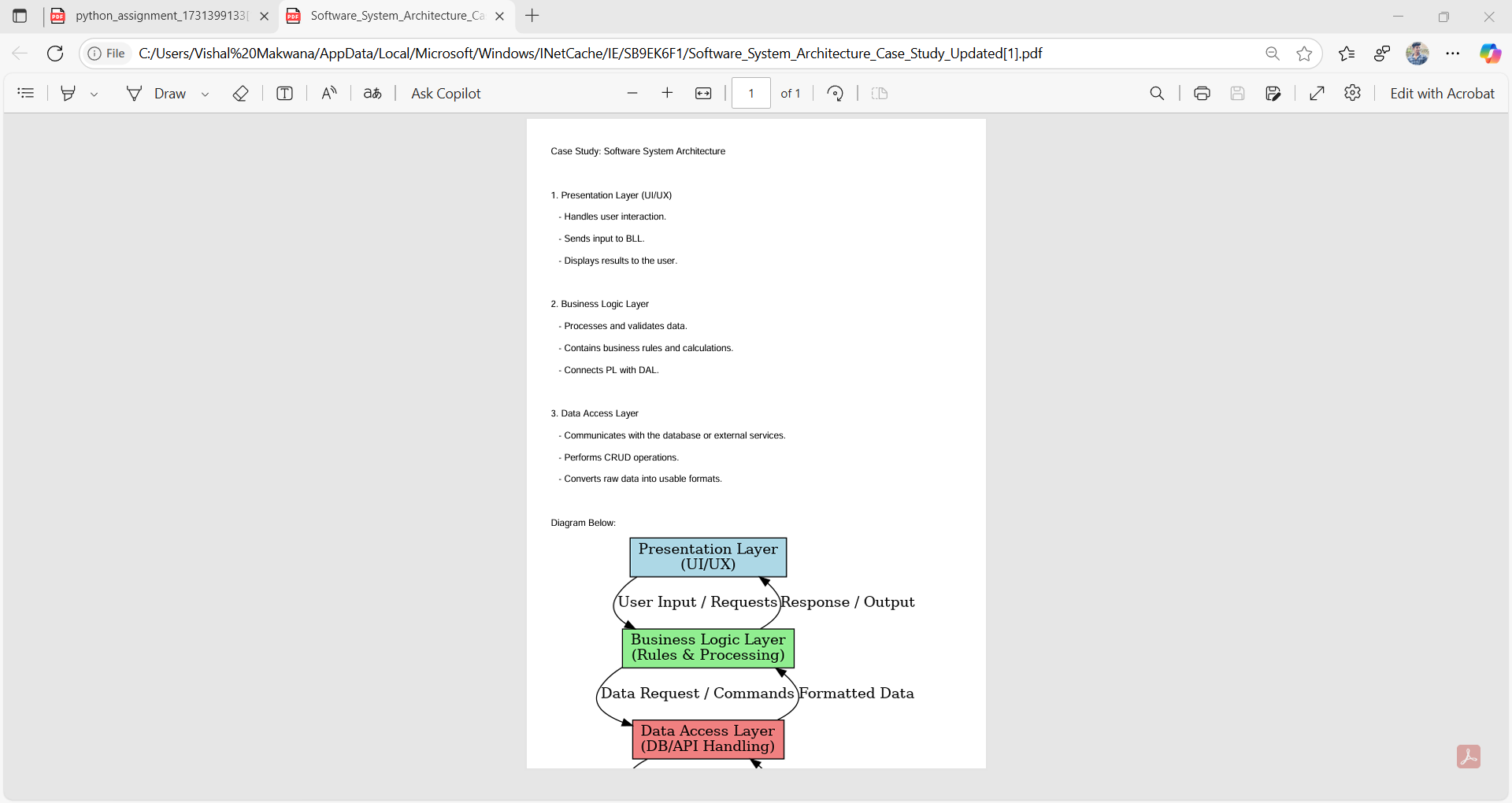
| **Software Name** | **Type** | **Classification** |
| --- | --- | --- |
| Microsoft Word | Word processing | Application Software |
| Google Chrome | Web browser | Application Software |
| Windows 10 (Operating System) | Operating system | System Software |
| File Explorer | File management tool | System Software |
| WhatsApp | Messaging application | Application Software |



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



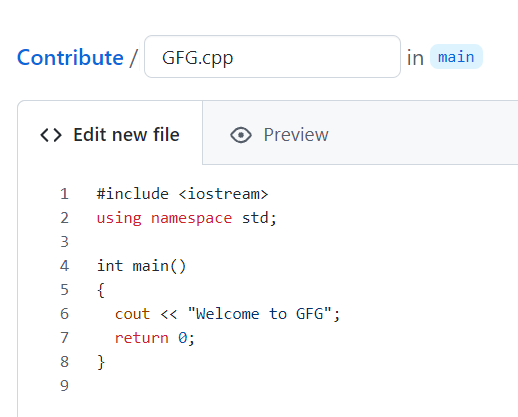
# 8) Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.



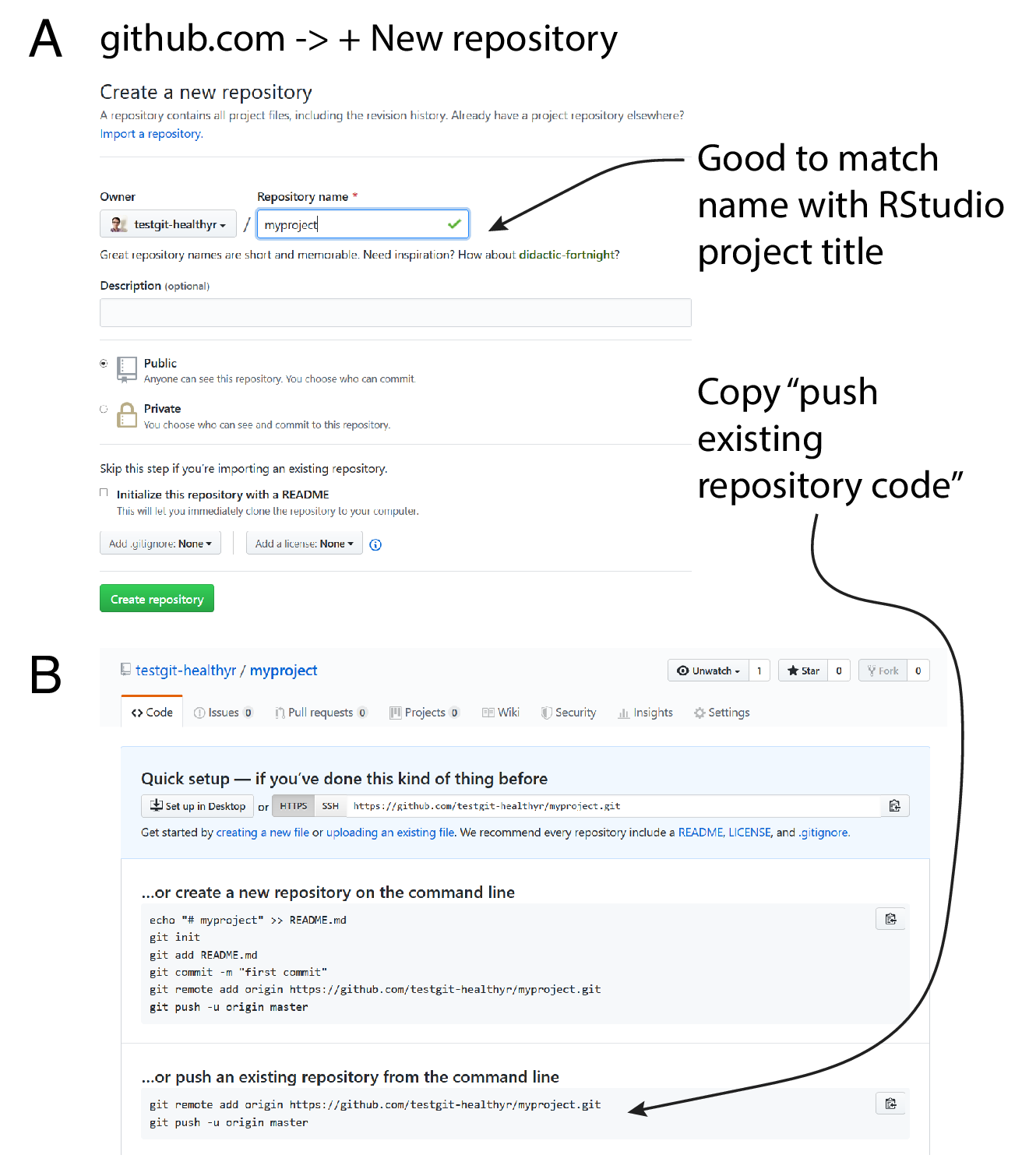
# 9) Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.

# 10) Write and upload your first source code file to Git hub.

* **Step 1: Write Your First Source Code File**
* **Step 2: Create a New GitHub Repository**
* **Step 3: Upload the File Using Git**



# 11)Create a Git hub repository and document how to commit and push code changes.



# 12)Create a student account on Git hub and collaborate on a small project with a classmate.

**Part 1: Create a GitHub Student Account**

1. **Visit** <https://github.com/join>
2. **Sign up** with your email, username, and password.
3. **Verify your email.**
4. **Apply for the GitHub Student Developer Pack**:
   * Go to: <https://education.github.com/pack>
   * Click “Get Student Pack.”
   * Use your **school email address** and provide proof of enrollment (e.g., student ID, school transcript, or a school-issued email).
   * Wait for approval (may take a few hours to a couple of days).

**🔹 Part 2: Set Up a Project Repository**

1. After account approval, **log in to GitHub**.
2. Click the **“+” icon** (top right) → **New repository**.
3. Name the repository (e.g., class-collab-project).
4. Select “Public” or “Private” and **initialize with a README**.
5. Click **Create repository**.

**🔹 Part 3: Collaborate with a Classmate**

1. On the repository page, go to **Settings → Collaborators**.
2. Add your classmate’s GitHub username or email to invite them.
3. Your classmate should accept the invitation from their notifications.

**🔹 Part 4: Work on the Project**

**1.Use Git or GitHub Desktop to clone the repository:**

**git clone** [**https://github.com/your-username/class-collab-project.git**](https://github.com/your-username/class-collab-project.git)

**2.Create or edit a file, e.g., hello.py:**

**print("Hello from Alice and Bob!")**

**3.Save and commit your changes:**

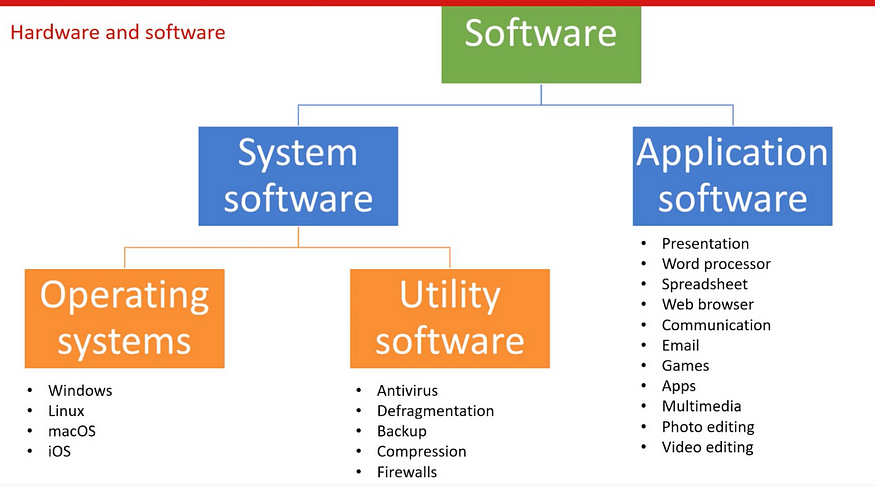
**git add hello.py**

**git commit -m "Added hello.py"**

**git push**

**4.Your classmate can now pull the changes, make edits, and push back.**

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



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

**✅ Step 1: Install Git**

Make sure Git is installed. Run this in your terminal:

* git –version

**✅ Step 2: Clone a Repository**

Find a public repo on GitHub or create your own. Then clone it:

* git clone https://github.com/username/repo-name.git
* cd repo-name

Or, create a local repo:

* mkdir git-practice
* cd git-practice
* git init

✅ Step 3: **Create a New Branch**

* git checkout -b feature-branch

Make some changes (e.g., add a file)

* echo "Hello Git" > hello.txt
* git add hello.txt
* git commit -m "Add hello.txt"

✅ Step 4: **Switch Back to Main Branch**

* git checkout main

Make a different change here too:

* echo "Main branch edit" > main.txt
* git add main.txt
* git commit -m "Add main.txt"

✅ Step 5: **Merge the Feature Branch**

* git merge feature-branch

✅ Step 6: **Push to Remote (Optional)**

* git push origin main

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

**1. Word Processing Software**

**Examples:** Microsoft Word, Google Docs, LibreOffice Writer  
**Use:** Creating, editing, formatting text documents  
**Productivity Impact:**

* Automates formatting and spelling correction
* Facilitates collaboration with features like comments and real-time editing
* Reduces time spent on document creation and revision

**2. Spreadsheet Software**

**Examples:** Microsoft Excel, Google Sheets  
**Use:** Data organization, calculations, financial analysis  
**Productivity Impact:**

* Automates complex calculations with formulas
* Supports data visualization with charts and graphs
* Enables forecasting and decision-making with built-in tools

**3. Presentation Software**

**Examples:** Microsoft PowerPoint, Google Slides, Prezi  
**Use:** Designing and presenting ideas visually  
**Productivity Impact:**

* Simplifies communication of concepts through visuals and animations
* Supports collaborative creation of presentations
* Saves time by using templates and design tools

**4. Database Management Software (DBMS)**

**Examples:** Microsoft Access, MySQL, Oracle Database  
**Use:** Storing, managing, and querying structured data  
**Productivity Impact:**

* Organizes large amounts of information efficiently
* Enhances data retrieval and reporting
* Reduces redundancy and ensures data integrity

**5. Communication Software**

**Examples:** Microsoft Teams, Zoom, Slack  
**Use:** Messaging, video conferencing, collaboration  
**Productivity Impact:**

* Reduces delays through instant messaging and quick meetings
* Facilitates remote work and teamwork across locations
* Integrates with other tools for seamless workflow

**6. Project Management Software**

**Examples:** Trello, Asana, Jira  
**Use:** Planning, scheduling, and tracking project progress  
**Productivity Impact:**

* Provides clear visibility into tasks and deadlines
* Improves time management and accountability
* Enhances coordination among team members

**7. Graphic Design and Multimedia Software**

**Examples:** Adobe Photoshop, Canva, AutoCAD  
**Use:** Creating visuals, animations, and design materials  
**Productivity Impact:**

* Speeds up content creation with templates and toolsets
* Allows creative professionals to produce high-quality outputs
* Integrates with publishing and marketing tools

**8. Web Browsers**

**Examples:** Google Chrome, Mozilla Firefox, Safari  
**Use:** Accessing web-based tools, information, and services  
**Productivity Impact:**

* Enables use of cloud-based applications
* Supports fast information retrieval and research
* Enhances workflow through browser extensions

## **15)Create a flowchart representing the Software Development Life Cycle (SDLC).**



**Stage-1: Planning and Requirement Analysis**

Planning is a crucial step in everything, just as in[software development](https://www.geeksforgeeks.org/software-development). In this same stage, [requirement analysis](https://www.geeksforgeeks.org/activities-involved-in-software-requirement-analysis)is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

**Stage-2: Defining Requirements**

In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.   
This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

**Stage-3: Designing Architecture**

[SRS](https://www.geeksforgeeks.org/software-requirement-specification-srs-format) is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.

**Stage-4: Developing Product**

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

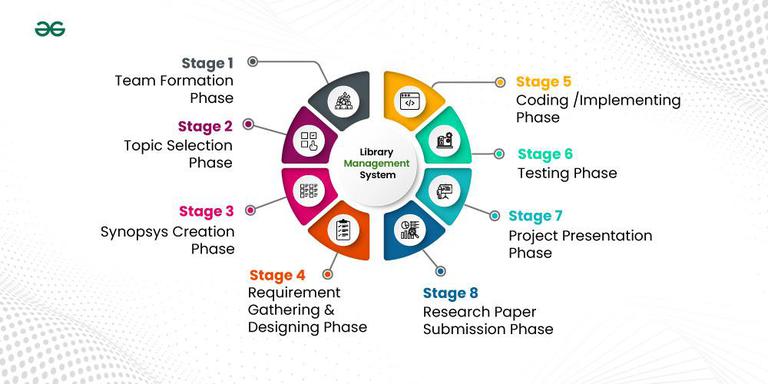
**Stage-5: Product Testing and Integration**

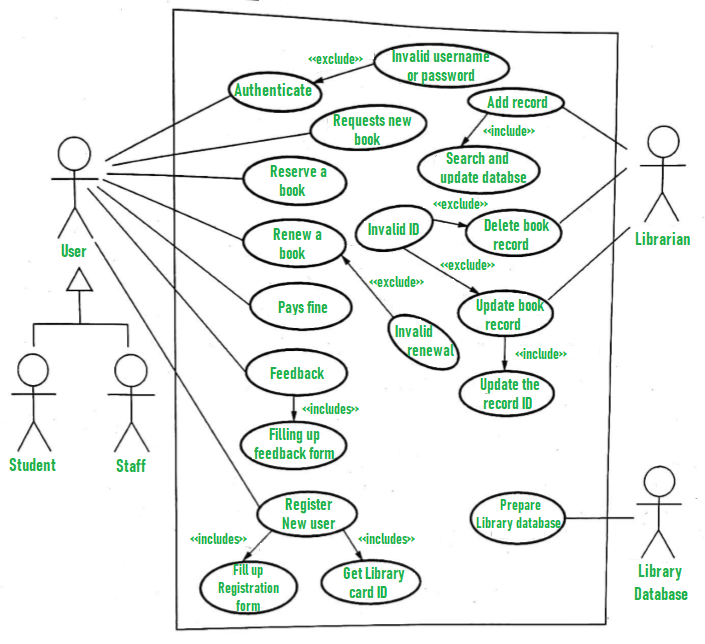
After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

**Stage-6: Deployment and Maintenance of Products**

After detailed testing, the conclusive product is released in phases as per the organization’s strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the [product’s supervision](https://www.geeksforgeeks.org/product-management).

## ***16)Write a requirement specification for a simple library management system.***





## **17)Perform a functional analysis for an online shopping system.**

**1. User Management**

**Functions:**

* **User Registration**
  + Collect user details (name, email, password, address, etc.)
* **User Login/Logout**
  + Secure authentication
* **Profile Management**
  + Update personal and shipping information
* **Password Recovery**
  + Email or OTP-based reset

**2. Product Catalog Management**

**Functions:**

* **Product Listing**
  + Display product name, image, price, category, and stock status
* **Search & Filter**
  + Search by keyword
  + Filter by price, brand, category, rating, etc.
* **Product Details**
  + Show description, specs, images, and reviews

**3. Shopping Cart System**

**Functions:**

* **Add to Cart**
  + Store selected items with quantity
* **View/Edit Cart**
  + Change item quantity or remove items
* **Price Calculation**
  + Show subtotal, tax, and total cost

**4. Checkout and Payment**

**Functions:**

* **Shipping Information**
  + Collect address and delivery method
* **Payment Integration**
  + Support payment gateways (e.g., credit card, PayPal)
* **Order Summary**
  + Final review before placing an order
* **Order Confirmation**
  + Send confirmation email/receipt

**5. Order Management**

**Functions:**

* **Order Tracking**
  + View current order status (pending, shipped, delivered)
* **Order History**
  + Users can view past purchases
* **Cancel/Return Requests**
  + Process cancellation or return requests per policy

**6. Review and Rating System**

**Functions:**

* **Post Reviews**
  + Allow users to write reviews for purchased products
* **Rate Products**
  + 1–5 star rating system

**7. Admin Panel (Back-End Management)**

**Functions:**

* **Product Management**
  + Add, update, or remove products
* **User Management**
  + View users, deactivate accounts if needed
* **Order Management**
  + View, process, and update order statuses
* **Analytics Dashboard**
  + Track sales, user activity, and inventory

**8. Security and Performance**

**Functional Considerations:**

* Encrypted login and payment data
* Rate-limiting or captcha to prevent abuse
* Responsive design for mobile and desktop

18)Design a basic system architecture for a food delivery app.

**🔷 1. Client Tier (Front-End)**

**a. User App (Customers)**

* Browse restaurants and menus
* Place orders
* Track delivery
* Make payments

**b. Restaurant App (Vendors)**

* Manage menu and availability
* Accept/reject orders
* Update order status

**c. Delivery App (Drivers)**

* Receive delivery assignments
* Navigate to pickup/drop-off
* Update delivery status

**🔷 2. Application Tier (Back-End Services / APIs)**

**a. User Management Service**

* Authentication (login/signup)
* Profile and address management

**b. Restaurant Management Service**

* Menu creation/editing
* Restaurant info, hours, ratings

**c. Order Management Service**

* Order placement, tracking, and status updates
* Assigning delivery partners

**d. Delivery Management Service**

* Real-time tracking of delivery agents
* Route optimization and ETA calculation

**e. Payment Service**

* Payment gateway integration
* Invoicing and refund processing

**f. Notification Service**

* In-app, SMS, or email notifications
* Order updates and promotions

**🔷 3. Data Tier (Database Systems)**

* **User DB:** Stores customer and driver data
* **Restaurant DB:** Stores menus, hours, ratings
* **Order DB:** Tracks order history and status
* **Payment DB:** Transaction records
* **Logs/Analytics DB:** User behavior, performance metrics

**🔷 4. Third-Party Integrations**

* **Maps API:** Google Maps or Mapbox for routing
* **Payment Gateway:** Stripe, PayPal, Razorpay
* **SMS/Email:** Twilio, SendGrid for communication

**🔷 5. Admin Panel (Web Interface)**

* Manage users, restaurants, and orders
* Monitor system activity
* View analytics and generate reports

**🔷 Deployment/Infrastructure**

* **Cloud Platform:** AWS, Azure, or GCP
* **Services:** Load balancer, Auto-scaling, CDN
* **Security:** HTTPS, data encryption, firewall

## **19)Develop test cases for a simple calculator program.**

#include <stdio.h>

int main() {

int a, b;

char op;

printf("Enter expression (e.g., 5 + 2): ");

scanf("%d %c %d", &a, &op, &b);

switch(op) {

case '+': printf("Result: %d\n", a + b); break;

case '-': printf("Result: %d\n", a - b); break;

case '\*': printf("Result: %d\n", a \* b); break;

case '/':

if (b == 0)

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

else

printf("Result: %d\n", a / b);

break;

default:

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

}

return 0;

}

## **20)Document a real-world case where a software application required critical maintenance.**

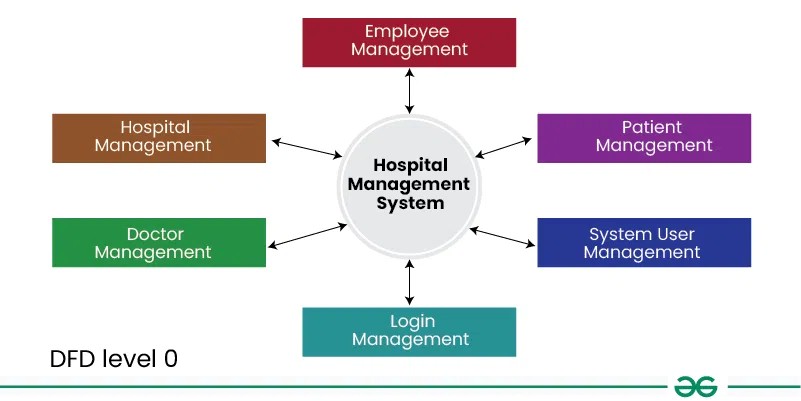
* **Problem:**

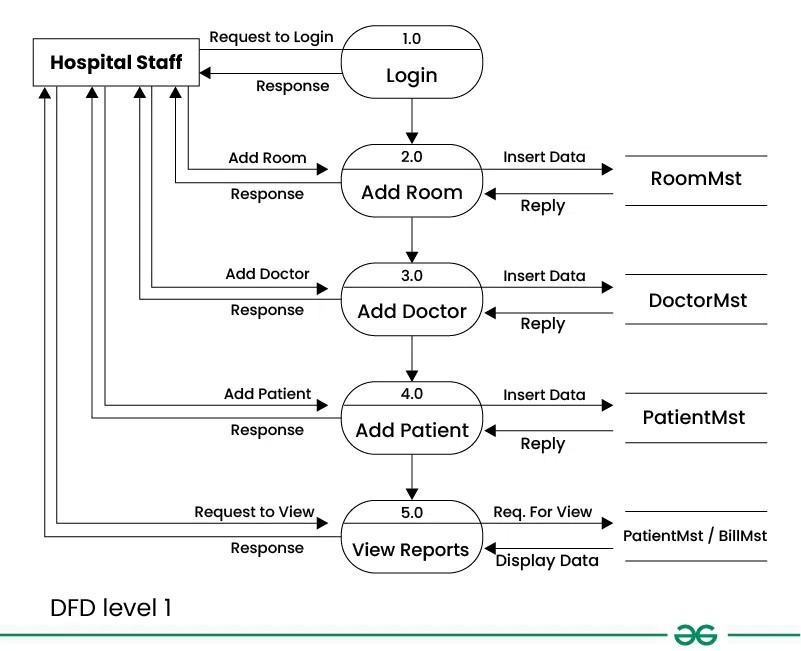
In 2017, a **large airline reservation system** suffered a critical failure that disrupted **flight bookings** and **check-ins** for several hours. The issue was caused by a **database failure** that led to the unavailability of real-time flight information, including booking and seat availability.

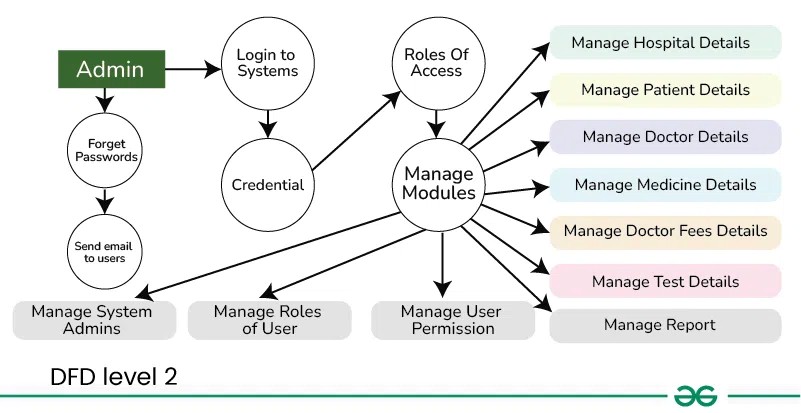
* **Impact:**
* **Flight bookings** were halted, preventing customers from making or changing reservations.
* **Check-in counters** became unresponsive, leading to delays and long lines at airports.
* **Customer dissatisfaction** soared as passengers were unable to confirm or modify their travel plans.
* **Revenue loss** due to missed bookings and cancellations.
* **Root Cause:**
* The failure was traced back to a **database crash** that resulted from a **lack of proper backup mechanisms** and inadequate **error handling** in the system.
* The system was not equipped with automatic failover or recovery procedures for such database failures.
* **Critical Maintenance Actions:**
* **Immediate Actions:**
  + The **database was restored** from backups after several hours of downtime.
  + A **manual override** was applied to allow limited flight bookings and check-ins while the main system was being restored.
* **Root Cause Analysis:**
  + The development and operations teams analyzed the incident and found that **backup protocols** were not regularly tested.
  + They identified the need for **automatic failover systems** to prevent similar outages in the future.
* **Long-Term Improvements:**
  + Implemented **automatic database failover** systems to ensure redundancy and minimize downtime.
  + **Improved backup procedures** to ensure frequent and reliable backups, along with regular recovery tests.
  + Introduced **error handling** mechanisms to detect and respond to database issues before they lead to failures.
* **Outcome:**
* The immediate issue was resolved, and the system was restored within a few hours.
* The airline's IT team implemented critical updates to improve **system reliability**, reducing the risk of similar failures in the future.
* **Customer trust** was partially regained by offering compensation to affected travelers and providing updates about the fixes.
* **Lessons Learned:**

1. **Redundancy is Key**: Implementing redundant systems like automatic failovers can ensure business continuity during a failure.
2. **Regular Backups and Testing**: Routine database backups and recovery tests can prevent extended downtimes in case of system crashes.
3. **Effective Error Handling**: Proper error detection and handling mechanisms should be in place to quickly mitigate issues before they impact users.

## **21)Create a DFD for a hospital management system**







## ***22)Build a simple desktop calculator application using a GUI library.***

import tkinter as tk

def on\_click(event):

text = event.widget.cget("text")

if text == "=":

try:

result = eval(str(entry.get()))

entry\_var.set(result)

except Exception as e:

entry\_var.set("Error")

elif text == "C":

entry\_var.set("")

else:

entry\_var.set(entry\_var.get() + text)

# Create main window

root = tk.Tk()

root.title("Simple Calculator")

root.geometry("300x400")

# Entry field

entry\_var = tk.StringVar()

entry = tk.Entry(root, textvar=entry\_var, font="Arial 20")

entry.pack(fill=tk.BOTH, ipadx=8, pady=10, padx=10)

# Button layout

buttons = [

["7", "8", "9", "/"],

["4", "5", "6", "\*"],

["1", "2", "3", "-"],

["0", ".", "=", "+"],

["C"]

]

for row in buttons:

frame = tk.Frame(root)

frame.pack(expand=True, fill="both")

for btn\_text in row:

btn = tk.Button(frame, text=btn\_text, font="Arial 18")

btn.pack(side="left", expand=True, fill="both")

btn.bind("<Button-1>", on\_click)

root.mainloop()

## **23)Draw a flowchart representing the logic of a basic online registration system.**

## 