**PHP Full Stack Assignment Set**

**Module 1 – Overview of IT Industry (LAB EXERCISE)**

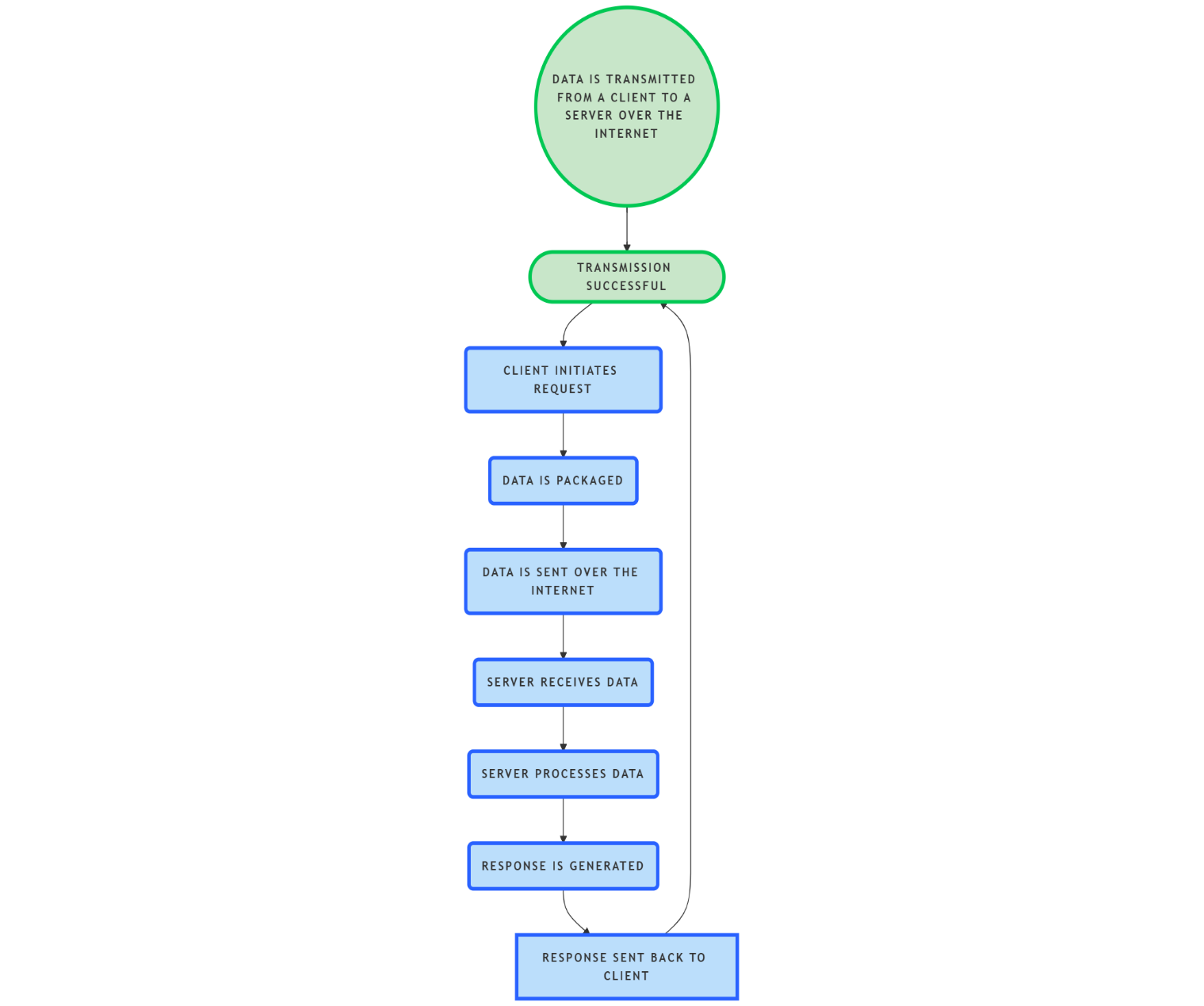
. Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

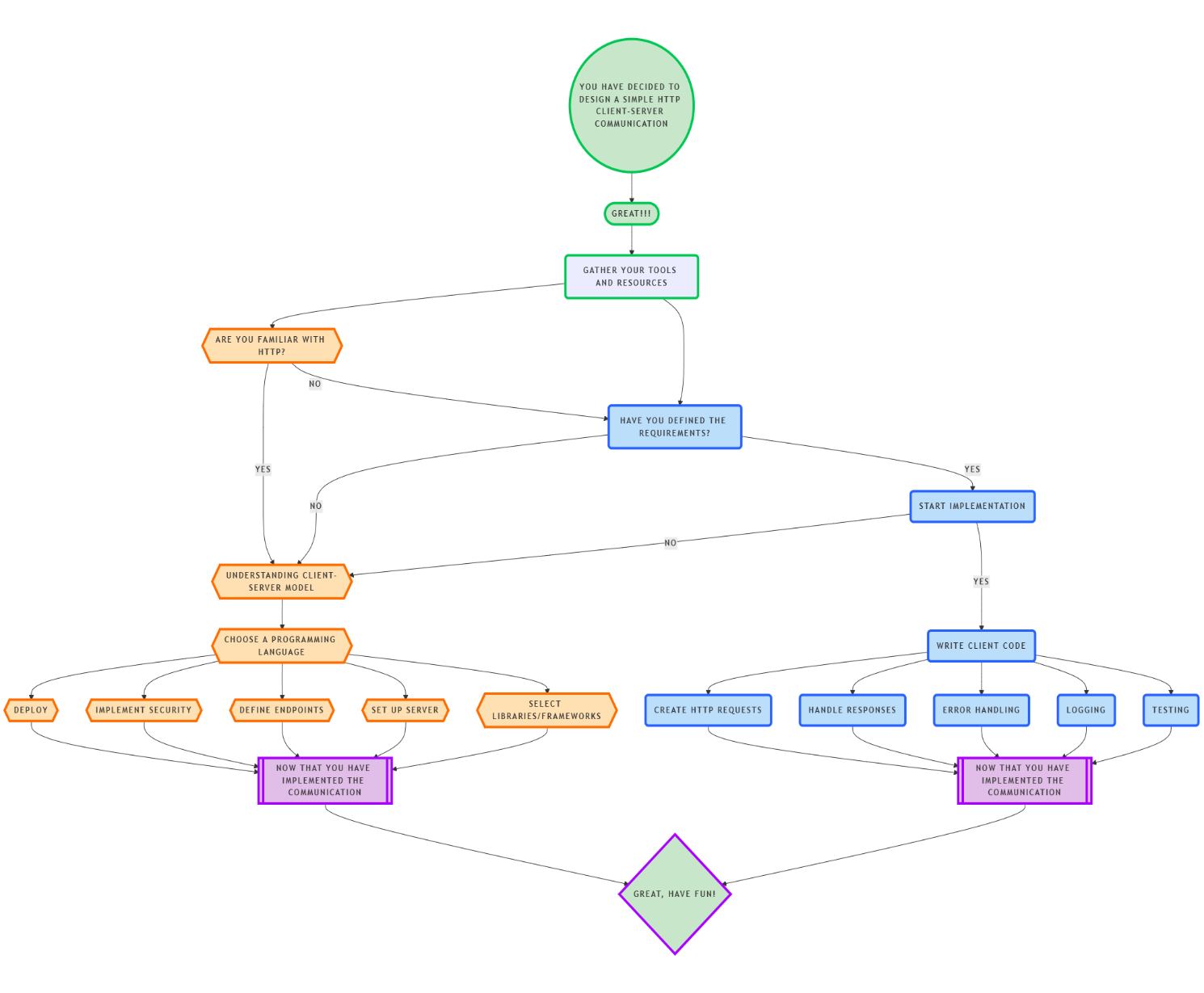
* #include<stdio.h>

main() {

printf(“\n Hello world”);

}

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

. simple HTTP client-server communication in any language.

**Protocols:**

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

* Example: Sending headers with the request

curl -H "User-Agent: MyCustomUserAgent" http://example.com

* Example: POST request

curl -X POST -d "username=myuser&password=mypassword" <http://example.com/login>

**Simulating an FTP Request:**

* Example: Downloading a file via FTP

curl -u username:password [ftp://example.com/path/to/file.txt -o downloaded\_file.txt](ftp://example.com/path/to/file.txt%20-o%20downloaded_file.txt)

* Example: Uploading a file via FTP

curl -u username:password -T local\_file.txt <ftp://example.com/path/to/upload/>

* Example: Listing files on an FTP server

curl -u username:password <ftp://example.com/path/to/>

* **Example: Get only the headers (HTTP response headers)**

curl -I <http://example.com>

**Application Security:**

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

* Ex. SQL Injection (SQLi)

Sql

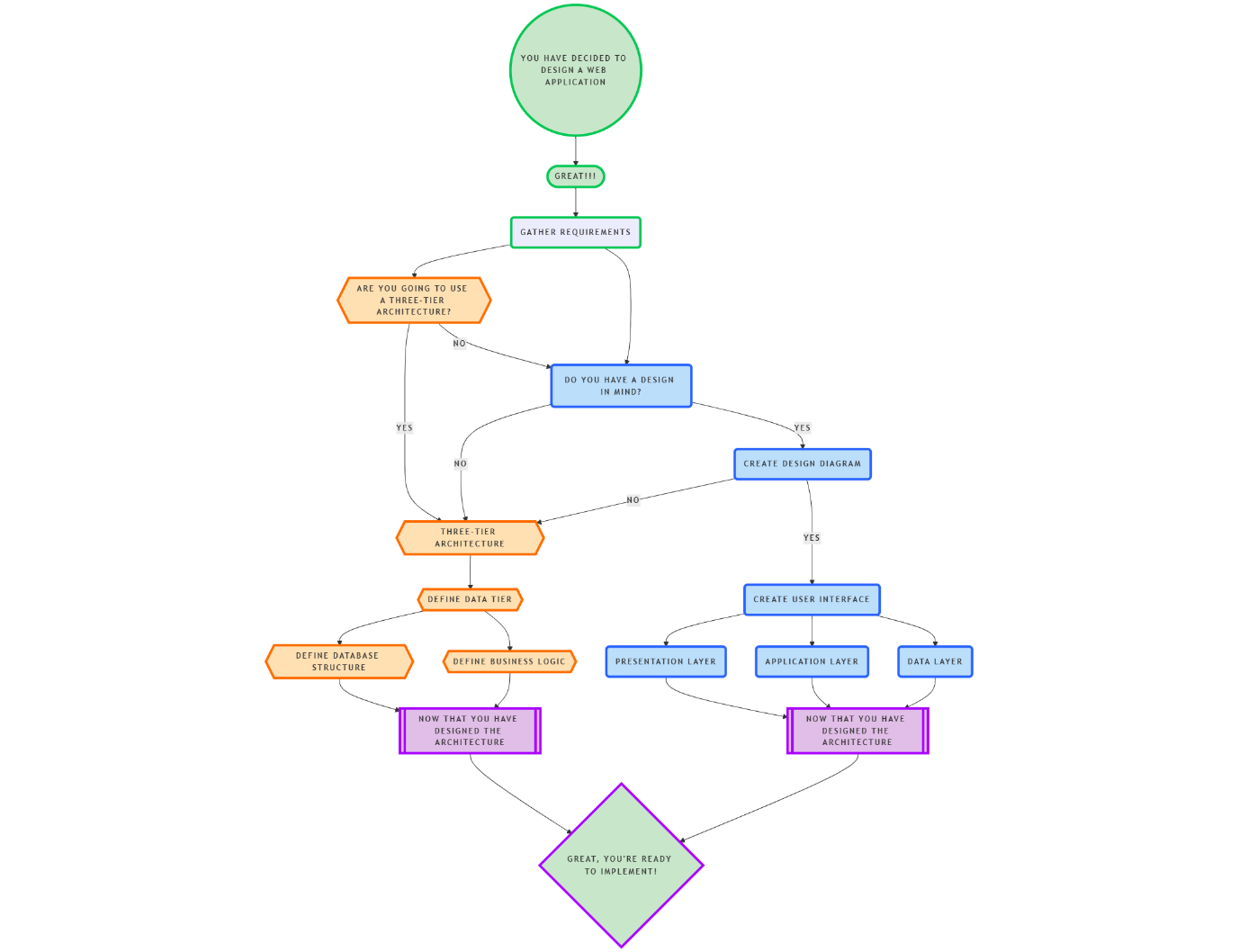
SELECT \* FROM users WHERE username = 'user' AND password = 'pass';

* Solution:

cursor.execute("SELECT \* FROM users WHERE username = %s AND password = %s", (username, password))

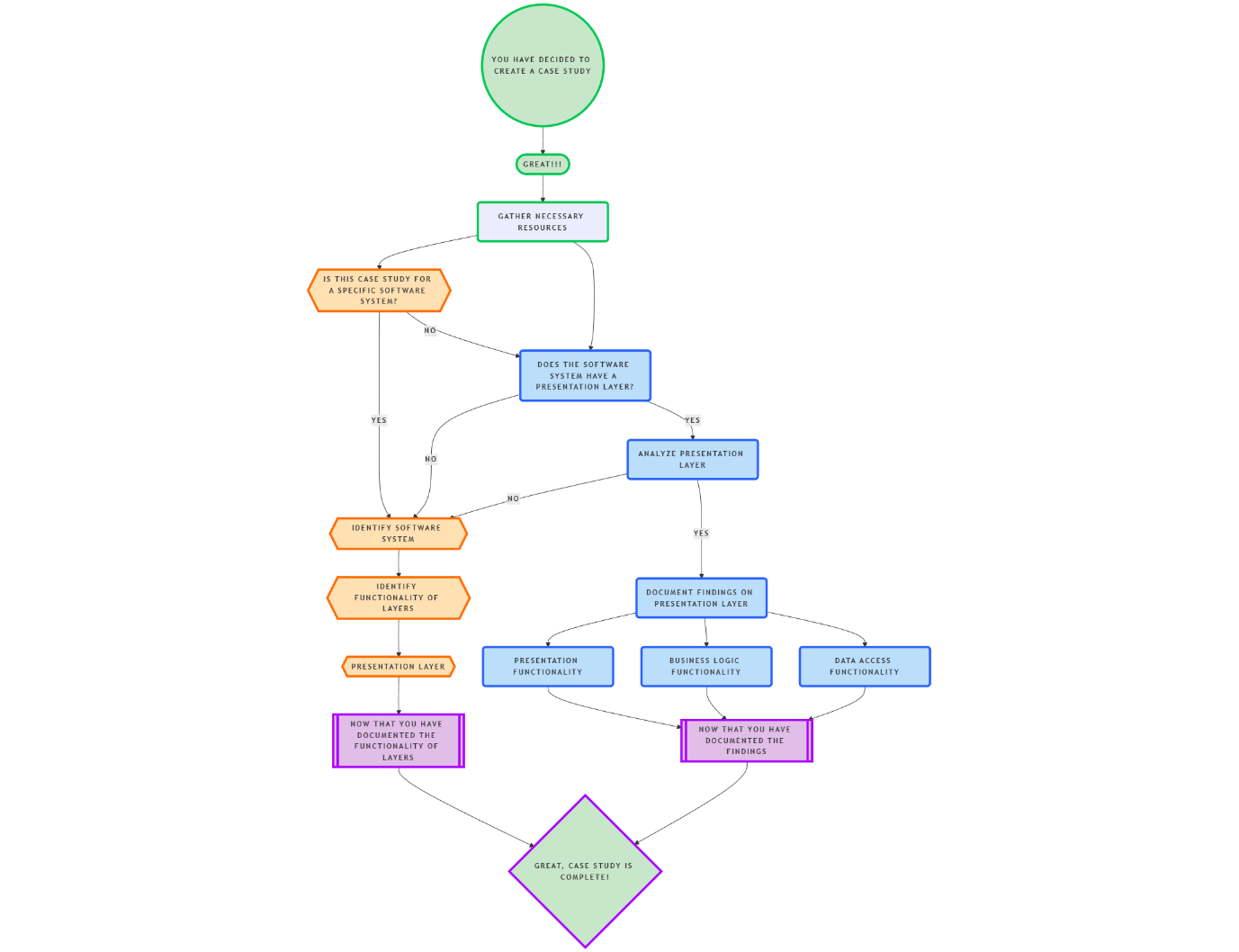
**Software Architecture:**

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



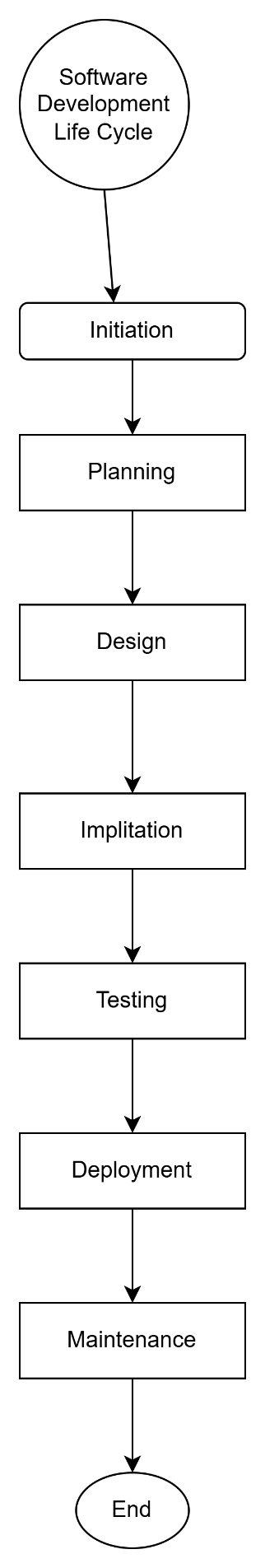
**Layers in Software Architecture:**

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



**Software Development Process:**

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



**Software Analysis:**

. Perform a functional analysis for an online shopping system.

**Functional Analysis for an Online Shopping System**

**1. Introduction**

An online shopping system is a platform that allows customers to browse, select, and purchase products or services over the internet. The system should provide an intuitive, secure, and reliable way for customers to shop online, while also offering tools for administrators to manage products, orders, payments, and customer data.

This functional analysis will focus on the key functionalities required for an online shopping system and categorize them into relevant modules that address both the customer and administrative requirements.

**2. Functional Modules of the Online Shopping System**

**2.1 User Registration and Authentication**

**Key Functions:**

* **User Registration**:
  + Users (customers) should be able to create an account by providing personal details such as name, email, password, phone number, and shipping address.
  + Users should have the ability to opt-in for newsletters, promotions, and offers.
* **User Authentication**:
  + Registered users should be able to log in using their email and password.
  + Password recovery options should be available for users who forget their login credentials.
* **Profile Management**:
  + Users should be able to update personal information such as contact details, shipping address, and payment preferences.

**Functionality Requirements:**

* Ensure secure authentication (use encryption and hashing for password storage).
* Implement multi-factor authentication for added security.
* Provide an easy-to-use password recovery system (email or SMS-based recovery).

**2.2 Product Catalog Management**

**Key Functions:**

* **Product Listings**:
  + Display a comprehensive list of available products categorized by type, brand, price, and other attributes (e.g., color, size).
* **Product Search**:
  + Users should be able to search for products based on keywords, categories, price range, or filters (e.g., size, color, ratings).
* **Product Detail Page**:
  + Each product will have a detail page showing its name, price, description, images, reviews, available sizes/colors, and related products.

**Functionality Requirements:**

* The system must support an easily navigable, filterable, and searchable product catalog.
* Product data (images, descriptions, etc.) should be dynamically updated to ensure accuracy.
* A responsive design should allow customers to view products on various devices (desktop, tablet, mobile).

**2.3 Shopping Cart Management**

**Key Functions:**

* **Add to Cart**:
  + Users should be able to add selected products to their shopping cart.
  + Users should be able to view the cart’s contents (product name, price, quantity, subtotal).
* **Cart Modification**:
  + Users should be able to modify quantities, remove items, or apply promo codes (discounts).
* **Cart Review**:
  + Before checkout, users should be able to review their cart, edit it, and see the final price including shipping costs, taxes, and discounts.

**Functionality Requirements:**

* The cart should be updated in real time as items are added or removed.
* Cart contents should be persistent (stored in cookies or a database) for a specific period to allow users to return to the cart at any time.

**2.4 Checkout Process**

**Key Functions:**

* **Shipping Information**:
  + The system should prompt users to enter their shipping address or select an existing address stored in their profile.
* **Payment Information**:
  + Users should be able to choose a payment method (credit card, debit card, PayPal, etc.) and securely enter payment details.
* **Order Summary**:
  + Display a final summary of the order with details such as shipping cost, tax, order total, and estimated delivery time.
* **Order Confirmation**:
  + Once payment is successful, users should receive an order confirmation page and an email receipt.

**Functionality Requirements:**

* Implement secure payment gateways (e.g., PayPal, Stripe, credit card processing).
* Provide order tracking features with updates on order status (shipped, delivered).
* Securely store payment data or integrate with third-party payment processors.

**2.5 Order Management**

**Key Functions:**

* **Order Processing**:
  + Admins can view and manage orders, update the order status (pending, shipped, delivered), and dispatch items.
* **Order Tracking**:
  + Customers can track the status of their orders via their account page, with updates such as shipping confirmation, tracking numbers, and delivery dates.
* **Order History**:
  + Customers can view a complete history of past orders and re-order previous items with ease.

**Functionality Requirements:**

* Real-time updating of order status.
* Provide a detailed breakdown of each order (items, delivery address, payment method, etc.).
* Support for refunds and returns with tracking of return status.

**2.6 Inventory Management**

**Key Functions:**

* **Stock Management**:
  + Admins can view current stock levels for each product, update inventory quantities, and mark items as out of stock.
* **Product Availability Alerts**:
  + Customers should be notified when an out-of-stock item becomes available.

**Functionality Requirements:**

* Ensure automatic updates for stock levels based on customer purchases and returns.
* Admins should be notified of low stock levels to trigger restocking.

**2.7 Reviews and Ratings**

**Key Functions:**

* **Product Ratings**:
  + Customers should be able to rate products on a scale (e.g., 1-5 stars) based on their experience.
* **Customer Reviews**:
  + Customers should be able to write product reviews, which should be visible to other users.

**Functionality Requirements:**

* Implement a moderation system to filter inappropriate reviews and ensure quality control.
* Display aggregated ratings (average stars) on the product detail page.

**2.8 Customer Support**

**Key Functions:**

* **Live Chat**:
  + A live chat feature where customers can directly interact with support representatives.
* **Help Center/FAQs**:
  + A knowledge base or help center with frequently asked questions to assist customers in solving common issues.
* **Return/Refund Request**:
  + An interface where customers can submit return or refund requests for faulty or unwanted products.

**Functionality Requirements:**

* Ensure the availability of customer support (via chat, email, phone) during business hours.
* Provide a user-friendly and informative FAQs section to reduce support load.

**3. Non-Functional Requirements**

**3.1 Performance and Scalability**

* The system should be able to handle thousands of concurrent users during peak shopping seasons (e.g., Black Friday or holiday sales).
* The platform should scale automatically to accommodate more products, users, and transactions.

**3.2 Security**

* Implement SSL (Secure Sockets Layer) encryption for secure transactions.
* Ensure secure storage of user data (e.g., passwords, payment details).
* Implement data protection protocols (e.g., GDPR compliance).

**3.3 Usability**

* The system should be intuitive, with easy-to-navigate interfaces for both customers and administrators.
* Provide mobile-friendly design to support users shopping on mobile devices.

**3.4 Availability**

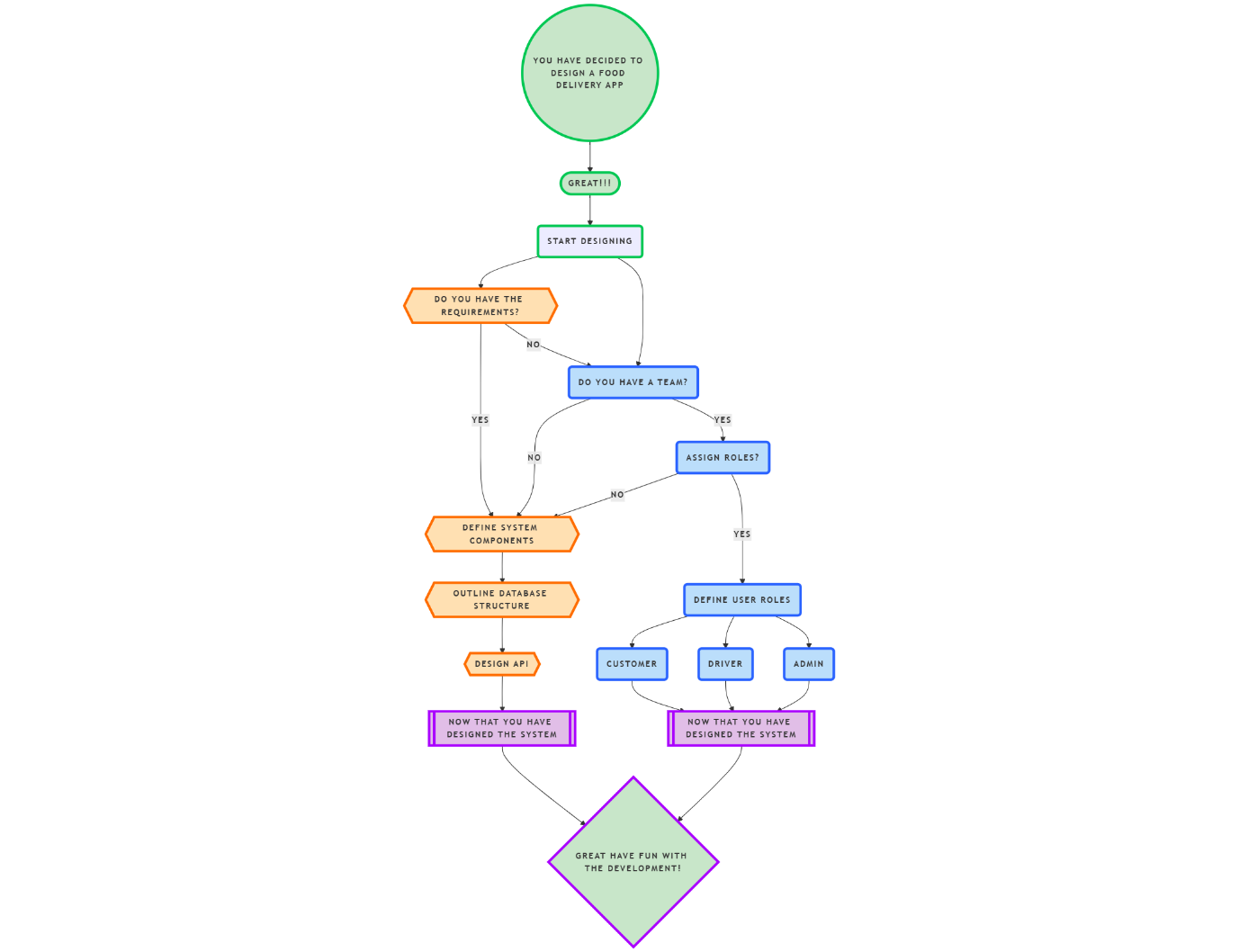
* The system should ensure high uptime, with minimal maintenance downtimes.
* Implement backups to ensure data recovery in case of a failure.

**3.5 Accessibility**

* Ensure the platform meets accessibility standards (WCAG) for users with disabilities, including screen reader compatibility and keyboard navigation.

**System Design:**

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



**Software Testing:**

. Develop test cases for a simple calculator program.

**Test Cases for a Simple Calculator Program**

A simple calculator program typically performs basic arithmetic operations like addition, subtraction, multiplication, and division. Below are the test cases that can be developed to validate the functionality of a basic calculator program.

**1. Test Case: Addition**

**Test Case ID: TC\_ADD\_001**

**Test Objective: Test the addition operation.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 5.
2. Select the addition operator +.
3. Enter the second number 3.
4. Press the = button.

**Expected Result: The output should be 8.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**2. Test Case: Subtraction**

**Test Case ID: TC\_SUB\_001**

**Test Objective: Test the subtraction operation.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 10.
2. Select the subtraction operator -.
3. Enter the second number 4.
4. Press the = button.

**Expected Result: The output should be 6.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**3. Test Case: Multiplication**

**Test Case ID: TC\_MUL\_001**

**Test Objective: Test the multiplication operation.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 6.
2. Select the multiplication operator \*.
3. Enter the second number 7.
4. Press the = button.

**Expected Result: The output should be 42.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**4. Test Case: Division**

**Test Case ID: TC\_DIV\_001**

**Test Objective: Test the division operation with non-zero divisor.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 20.
2. Select the division operator /.
3. Enter the second number 5.
4. Press the = button.

**Expected Result: The output should be 4.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**5. Test Case: Division by Zero**

**Test Case ID: TC\_DIV\_002**

**Test Objective: Test division by zero, which should raise an error.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 10.
2. Select the division operator /.
3. Enter the second number 0.
4. Press the = button.

**Expected Result: The output should display an error message (e.g., "Cannot divide by zero" or "Error").**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**6. Test Case: Input of Negative Numbers**

**Test Case ID: TC\_NEG\_001**

**Test Objective: Test the calculator with negative numbers in an operation.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number -8.
2. Select the addition operator +.
3. Enter the second number 5.
4. Press the = button.

**Expected Result: The output should be -3.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**7. Test Case: Floating Point Operations**

**Test Case ID: TC\_FLOAT\_001**

**Test Objective: Test the calculator's handling of floating point numbers.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 3.5.
2. Select the multiplication operator \*.
3. Enter the second number 2.2.
4. Press the = button.

**Expected Result: The output should be 7.7.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**8. Test Case: Multiple Operations**

**Test Case ID: TC\_MULTI\_001**

**Test Objective: Test performing multiple operations in sequence.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 5.
2. Select the addition operator +.
3. Enter the second number 3.
4. Press the = button to get the result 8.
5. Now select the multiplication operator \*.
6. Enter the third number 2.
7. Press the = button.

**Expected Result: The output should be 16.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**9. Test Case: Clear Button**

**Test Case ID: TC\_CLR\_001**

**Test Objective: Test the "Clear" button functionality.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the number 12.
2. Press the "Clear" button.

**Expected Result: The screen should reset to 0 or blank.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**10. Test Case: Input of Large Numbers**

**Test Case ID: TC\_LG\_NUM\_001**

**Test Objective: Test the calculator's handling of large numbers.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 999999.
2. Select the multiplication operator \*.
3. Enter the second number 999999.
4. Press the = button.

**Expected Result: The output should be 999998000001.**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**11. Test Case: Decimal Precision**

**Test Case ID: TC\_PREC\_001**

**Test Objective: Test the calculator's precision with decimals.**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter the first number 1.3333.
2. Select the addition operator +.
3. Enter the second number 0.6667.
4. Press the = button.

**Expected Result: The output should be 2.0000 (rounded to four decimal places).**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**12. Test Case: Invalid Input Handling**

**Test Case ID: TC\_INVALID\_001**

**Test Objective: Test the calculator’s handling of invalid inputs (e.g., letters or special characters).**

**Pre-Conditions: Calculator program is open and operational.**

**Test Steps:**

1. Enter a non-numeric character a.
2. Press the = button.

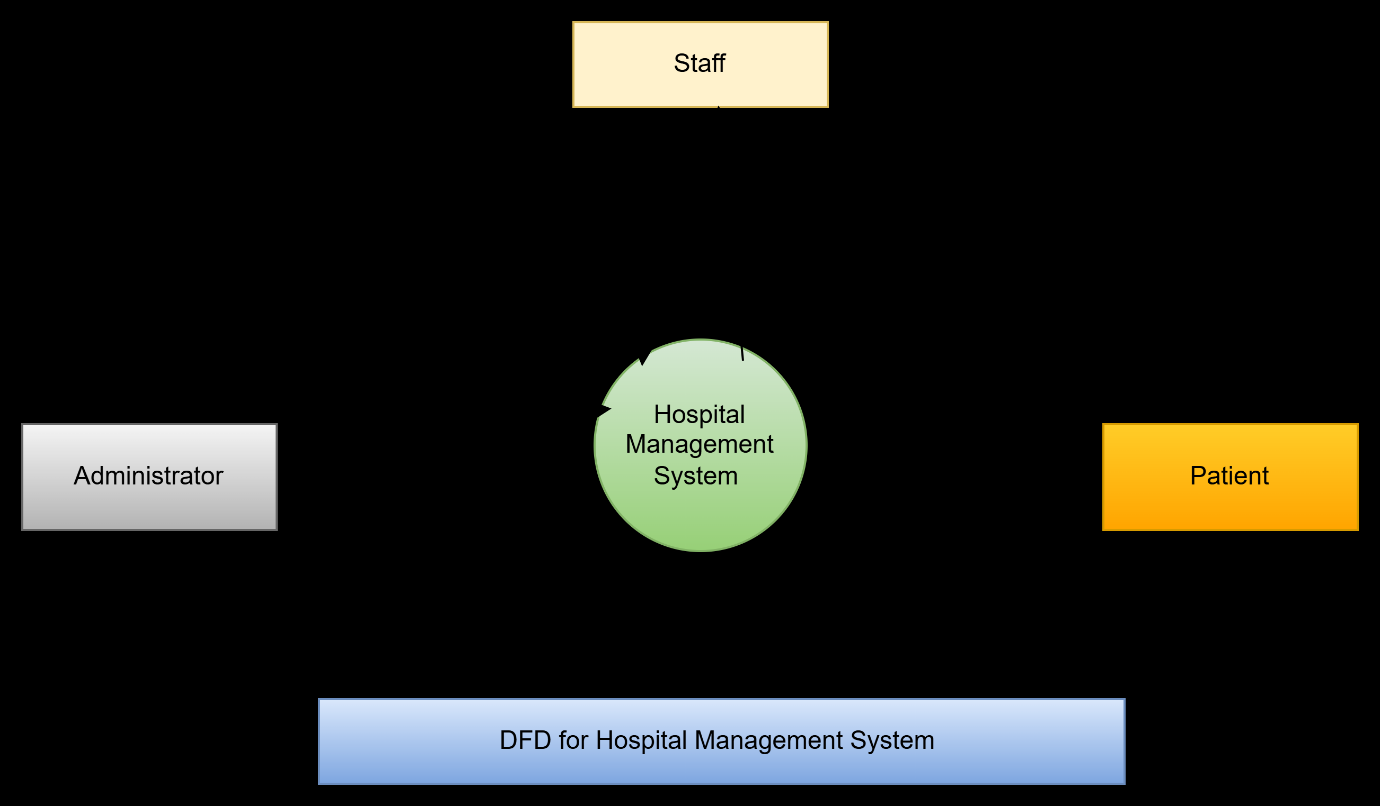
**Expected Result: The output should display an error message (e.g., "Invalid input").**

**Actual Result: (To be filled after testing)**

**Status: (Pass/Fail)**

**DFD (Data Flow Diagram):**

. Create a DFD for a hospital management system.



**Flow Chart:**

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

