Theory exercises

1) explain in your own words what a program is and how it functions.what is programming

A) a computer programming is a set of instructions that tell a computer how to perform a specific task. it's essentially a recipe for the computer, guiding it through a series of steps to achieve a desired outcome. these instructions are written in a programming language, which is a formal language that a computer can understand

2) what are the key steps involved in the programming process? types of programming language

A) the programming language also known as the software development life cycle, generally involves these key steps:

problem analysis, design, coding, testing and debugging and documentation. some also include implementation, deployment and maintainance.

3) what are the main difference between high-level and low-level programming languages?

A) high-level languages are designed to be human readable, easier to learn and more portable, while low-level languages offer more control over hardware and potentially higher performance but are more complex and machine-dependent.

4) describes the roles of the client and server in web communication.

A) in a client-server network, the client initiates requests for resources or services such as web browser, while the server provides those resources and services such as web server.

5) explain the function of the TCP/IP modle and its layers

A) the four layers of the TCP/IP model and their functions:

application(SMTP,HTTP/HTTPS,FTP,SSH)----generates the data, requests the connections

transport(UDP,TCP)----establishes an error free data connection, splits the data into smaller packets, obtais acknowledgement of the reception of the packets.

internet(IPv4/IPv6,ICMP,ARP)----sends the packets,ensures that the packets are sent accurately, routes data to the correct network.

network accsess---- adds the destination on MAC address, sends data between applications over the network, handles the physical infrastructure.

6) explain client-server communication

A) it is a distributed communication model where clients request services or resources from a server, and the server responds to those requests

7) how does broadband differ from fibre-optic internet?

A) brosdband is a general term for high-speed internet access, while fibre optic internet is a specific type of broadband technology that uses light signals transmitted through thin glass or plastic fibres. fibre optic internet offers significantly faster speeds and greater reliability than other forms of broadband like DSL or cable,

which use electrical signals transmitted through copper wires.

8) what are the difference between HTTP and HTTPS protocols?

A) the main difference between HTTP(hypertext transfer protocol) transmits data in plain text, making it vulnerable to eavesdropping and tampering. HTTPS(hypertext transfer protocol secure)encrypts data using SSL/TLS, providing a secure connection and protecting sensitive information

9) what is the role of encryption in securing application?

A) encryption is vital for securing applications by protecting sensitive data both at rest(stored) and in transit(being transmitted). it converts readable data into an unreadable format, making it unusable to unauthorized individuals even if they gain access to it. this protects data from theft , tampering and unauthorized access, ensuring confidentiality and integrity.

10) what is the difference between system software and application software?

A) System software manages the computer's hardware and provides a platform for other software to run, while application software allows users to perform specific tasks.

11) what is the significance of modularity in software architecture?

A) Modularity in software architecture is significant because it breaks down complex systems into smaller, manageable, and independent modules or components. This approach enhances various aspects of the software development lifecycle, including design, development, testing, and maintenance.

12) why are layers important in software architecture?

A) layers in software architecture are crucial for building robust, maintainable and scalable applications. They promote modularity, seperation of concerns and easier development by dividing the system into distinct, managable units with specific responsibility.

13) Explain the importance of a development environment in software production.

A) a development enviroment is crucial in software production as it provides a dedicated space for developers to build, test, and refine software before it's released to users. It allows for controlled experimentation, error detection and iterative improvement, ultimately leading to higher quality software and increased developer productivity.

14) what is the difference between source code and machine code?

A) Source code is human-readable instructions for a computer, written in a programming language. Machine code is the binary code that a computer's CPU directly executes. Source code must be translated (compiled or interpreted) into machine code for the computer to understand and run it.

15) why is version control is important in software

development?

A) because it enables teams to track changes to code, collaborate efficiently and maintain a reliable history of the project. This leads to the better organization, reduced errors and faster development cycles

16) What are the benefits of using Github for students?

A) github offers several key benefits for students primarily through the github student developer pack. This pack provides access to premium tools, cloud services and learning resources, enabling students to collaborate on projects, build a portfolio and develop essential technical skills.

17) What are the differences between open-source and proprietary software?

A) open-source and prioriety software differ primarily in terms of ownership, access to source code and licensing terms . open source software has publicy available source code, allowing users to modify, distribute and use it freely, often with no cost. Proprietary software, on the other hand, is owned by a specific individual or company, with restricted access to the source code and specific licensing terms for its use.

18) How does GIT improve collaboration in a software development team?

A) git improves collaboration in software development teams by providing a decentralized version control system that allows multiple developers to work on the same project simultaneously without conflicts. Features like branching, merging and pull

requests, faciliate parallel development, code review and integration of changes.

19) what is the role of application software in business?

A) Application software plays a vital role in businesses by providing the tools to streamline operations, enhance productivity, and improve decision-making. It enables businesses to manage various tasks, from basic word processing and data analysis to complex functions like customer relationship management (CRM) and enterprise resource planning (ERP). Ultimately, it helps businesses become more efficient, competative and profitable.

20) What are the main stages of the software development process?

A) the software development process, also known as the software development life cycle (SDLC), typically involves several key stages. These include planning, requirements analysis, design, development(or coding), testing, deployment and maintainance. Each stage plays a crucial role in ensuring the successful creation and delivery of software.

21) why is the requirement analysis phase critical in software development?

A) the requirement analysis phase is critical in software development because it lays the foundation for a successful project by ensuring that the software being built meets the needs of stakeholder and users. It involves identifying, documenting and analyzing requirements to avoid costly errors and delays later in the development process.

22) What is the role of software analysis in the development process?

A) Software analysis plays a crucial role in the development process by ensuring that the software being built meets the needs of stakeholders and functions effectively. It involves understanding the requirements of the software, analyzing them, and designing a solution that addresses those requirements. This phase is essential for preventing costly mistakes, reducing risks, and ultimately delivering a successful software product.

23) What are the key elements of system design?

A) Key elements of system design include architecture, data flow, scalability, performance, security, and reliability. These elements work together to ensure a robust, efficient, and maintainable system.

24) Why is software testing important?

A) Software testing is crucial for ensuring the reliability, security, and performance of software applications, leading to higher customer satisfaction and reduced costs. By identifying and fixing bugs early in the development process, testing minimizes the risk of costly errors and ensures a more robust and user-friendly product.

25) What types of software maintenance are there?

A) There are four main types of software maintenance: corrective, adaptive, perfective, and preventive. These categories cover different aspects of maintaining software after its initial release, ensuring it continues to function correctly, meets evolving needs and remains efficient.

26) What are the key differences between web and desktop applications?

A) Web and desktop applications differ primarily in how they are accessed and deployed. Web applications run within a web browser, making them accessible from any device with an internet connection, while desktop applications are installed and run directly on a specific computer, offering potential performance advantages.

28) What are the advantages of using web applications over desktop applications?

A) Web applications offer several advantages over desktop applications, primarily due to their accessibility, ease of updates, and platform independence. Web apps can be accessed from any device with a web browser and internet connection, eliminating the need for installation and offering greater flexibility for users working remotely or on different devices. Centralized updates mean users always have the latest version without manual intervention, and the platform independence means they work across different operating systems.

29) What role does UI/UX design play in application development?

A) UI/UX design plays a crucial role in application development by ensuring applications are user-friendly, visually appealing, and effective, ultimately driving user satisfaction and app success. It acts as the backbone of an application, influencing its structure, navigation, and overall user experience.

30) What are the differences between native and hybrid mobile apps?

A) Native and hybrid mobile apps differ primarily in their development approach, performance, user experience, and cost. Native apps are built specifically for a single platform (iOS or Android) using platform-specific languages, offering superior performance and access to device features. Hybrid apps, on the other hand, are built with web technologies (HTML, CSS, JavaScript) and wrapped in a native container, allowing cross-platform compatibility but potentially sacrificing some performance and native feature access.

31) What is the significance of DFDs in system analysis?

A) data flow diagrams (DFDs) are crucial in system analysis because they visually represent how data moves through a system, aiding in understanding, communication and identifying potential issues. They help in clarifying system processes, identifying areas for improvement, and ensuring that the final system meets its intended design.

32) what are the pros and cons of desktop applications compared to web applications ?

A) desktop applications generally offer better performance and offline functionality while web applications excel in accessibility, ease of updates and cross-platform compatibility he best choice depends on specific needs, including performance requirements, user location, and the need for real-time collaboration.

33) How do flowcharts help in programming and system design?

A) Flowcharts provide a visual representation of the logic and steps within a program or system, making them valuable tools for both programming and system design. They help in planning, understanding, and communicating complex processes, improving efficiency and reducing errors.

Lab exercises

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

A)

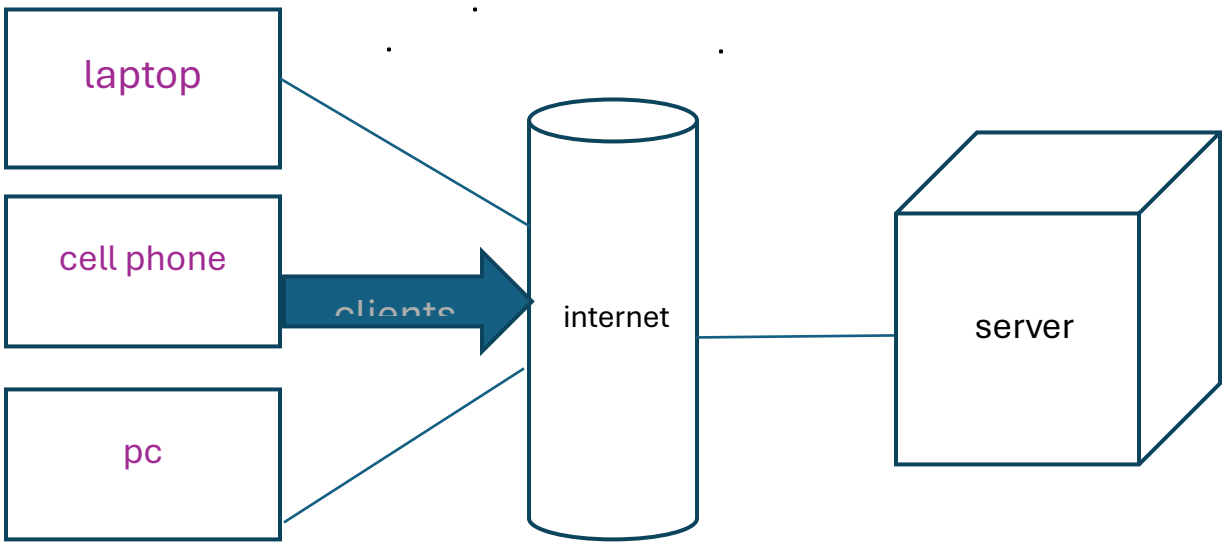
Python
print("Hello, World!")

B)

```java
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}
```

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

A) in a client-server model, data transmission over the internet involves a client initiating a request, which is then processed by a server, and a response being sent back to the client. This interaction is facilitated by networking protocols like TCP/IP and application protocols like HTTP. The process can be visualized as follows:



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

A) python

```python
# server.py
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print(f"Server serving at port {PORT}")
    httpd.serve_forever()

# client.py
import http.client

host = "localhost"
port = 8000

# Establish connection
conn = http.client.HTTPConnection(host, port)

# Make a GET request
conn.request("GET", "/")

# Get the response
response = conn.getresponse()

# Print status and response body
print(f"Status: {response.status}, Reason: {response.reason}")
print(response.read().decode())

# Close the connection
conn.close()
```

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

A) Different types of internet connections offer varying speeds, reliability, and costs. Broadband, including fibre and cable, generally provides faster speeds and lower latency, while satellite internet is better for rular areas but may have higher latency and data limitations:

Broadband(cable/DSL):

Pros: widely available, especially in urban and suburban areas, offers decent speeds, and is generally more affordable than fibre. Cable connections can be faster than DSL.

Cons: speeds can slow down during peak hours due to bandwidth sharing and latency can be higher than fibre.

Fibre optic:

Pros: extremely fast speeds(both download and upload), very low latency and highly reliable.

Cons: more expensive than other options, availability is limited to areas with fibre infrastructure and installatiion can be more complex.

Satellite internet:

Pros: available in remote and rular areas where other options aren't available, can be a good option for those with no other internet access

Cons: higher latency(delay in data transmission), can be affected by weather conditions (rain fade), and often has data caps and higher costs compared to wired connections.

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

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

Simulating HTTP and FTP requests using curl

HTTP requests

1. GET requests

To simulate a basic GET request, which retrieves data from a specified URL, simply type curl followed by the URL:

bash

curl https://www.example.com

Use code with caution.

This will output the entire HTML content of the webpage to your terminal.

Fetching only HTTP headers

To view only the HTTP headers without the response body, use the -I option:

bash

curl -I https://www.example.com

Use code with caution.

This sends an HTTP HEAD request, which asks the server to send only the headers.

Sending parameters

Include parameters in GET requests by adding them to the URL or by using the -G and -d options:

bash

curl -G -d "param1=value1" -d "param2=value2" http://httpbin.org/get

Use code with caution.

Including custom headers

Use the -H option to send custom headers with the request:

bash

curl -H "User-Agent: MyCustomBrowser/1.0" https://www.example.com

Use code with caution.

2. POST requests

To simulate a POST request, which is used to send data to a server, use the -X POST option along with the data to be sent using the -d option:

bash

curl -X POST -H "Content-Type: application/json" -d '{"key": "value"}' https://educative.com/api/login

Use code with caution.

This example sends a JSON payload to the server.

Sending form data

You can also send form data using the -d option in application/x-www-form-urlencoded format:

bash

curl -X POST https://educative.com/api/submit-form -H "Content-Type: application/x-www-form-urlencoded" -d "username=arya&password=got"

Use code with caution.

3. PUT requests

The PUT method creates a new resource or replaces an existing one on the server with the request payload.

bash

curl -X PUT https://reqbin.com/echo/put/json -H "Content-Type: application/json" -d '{"key": "value"}'

Use code with caution.

This example sends JSON data to create or update a resource on a server, according to ReqBin.

FTP requests

1. Downloading files

Download a file from an FTP server using curl by specifying the FTP URL and providing authentication details if required:

bash

curl --user username:password -o filename.tar.gz ftp://domain.com/directory/filename.tar.gz

Use code with caution.

--user username:password: provides authentication credentials.

-o filename.tar.gz: specifies the output filename for the downloaded file.

2. Uploading files

To upload a file to an FTP server, use the -T option to specify the local file path:

bash

curl -T local_file_path ftp://ftp.server.com/remote_directory/

Use code with caution.

local_file_path: is the path to the file you want to upload.

ftp://ftp.server.com/remote_directory/: specifies the FTP server and the target directory for the upload.

Uploading with authentication

For FTP uploads requiring authentication, include the --user option with credentials:

bash

curl -u username:password -T local_file_path ftp://ftp.server.com/remote_directory/

Use code with caution.

Important considerations

Replace placeholders like https://www.example.com, username, password, filename.tar.gz, local_file_path, ftp.server.com, and remote_directory with the appropriate values for specific scenarios.

Be cautious about including sensitive information like passwords directly in the command line, especially in shared or logged environments. Consider using environment variables or other secure methods for handling sensitive data.

Explore curl's extensive documentation and options for advanced functionality, including handling redirects, cookies, proxies, and error handling.

Lab exercise 6: Identify and explain three common application security vulnerabilities. Suggest possible solutions.

Three common application security vulnerabilities and solutions

Here are three common application security vulnerabilities, their explanations, and potential solutions:

1. Injection flaws (including SQL injection and cross-site scripting (XSS))

Explanation: Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. SQL injection (SQLi) allows attackers to manipulate a web application's database by injecting malicious SQL code through user inputs. XSS vulnerabilities enable attackers to inject malicious scripts (often JavaScript) into web pages, which are then executed in the victim's browser.

Impact: Attackers can gain unauthorized access to sensitive data (like user credentials or personal information), modify or delete data, take over user sessions, and potentially execute malicious code on the server or in the user's browser.

Solutions:

SQL : Use prepared statements and parameterized queries to separate data from code. Validate and sanitize all user inputs to ensure they conform to expected formats and don't contain malicious SQL. Implement the principle of least privilege for database users, limiting access to only what's necessary.

XSS: Encode user inputs before displaying them in HTML, JavaScript, or other contexts, preventing malicious scripts from executing. Legit Security suggests avoiding returning HTML tags to the client to prevent XSS attacks. Use a strong Content Security Policy (CSP) to restrict script execution to trusted sources.

2. Broken authentication

Explanation: Broken authentication involves flaws in how an application manages user identification, authentication, and session management, allowing attackers to bypass login systems or hijack user accounts. This can arise from weak password policies, poor session management, insecure password recovery processes, and a lack of multi-factor authentication.

Impact: Attackers can gain unauthorized access to user accounts, steal credentials, escalate privileges, and potentially compromise an entire system.

Solutions: Implement strong password policies that require complex passwords and discourage reuse. Employ multi-factor authentication (MFA) to require additional verification beyond just a password. Secure session management practices are crucial, including using secure session tokens, invalidating sessions after logout, and implementing timeouts. Regularly audit authentication and session management for weaknesses.

3. Sensitive data exposure

Explanation: Sensitive data exposure happens when applications fail to adequately protect critical information like passwords, credit card numbers, or personal data. This can occur when data is transmitted in plain text, stored without proper encryption, or exposed in URLs, logs, or error messages.

Impact: Attackers can intercept, steal, or misuse sensitive data, potentially leading to identity theft, financial fraud, and compliance violations.

Solutions: Encrypt data both at rest and in transit using strong encryption algorithms like AES-256. Use secure protocols like HTTPS with proper certificates for data transmission. Avoid storing sensitive data unnecessarily, and implement data masking in non-production environments. Regularly audit data access logs for unusual patterns and ensure compliance with data protection regulations like GDPR, PCI DSS or HIPAA.

SQL injection:

What it is:

SQL Injection occurs when an attacker injects malicious SQL code into an application's input fields, which are then used to construct database queries. This allows the attacker to manipulate the database, potentially accessing, modifying, or deleting data without authorization.

Example:

An attacker could input a specially crafted string into a username field that, when combined with the database query, allows them to bypass authentication or retrieve all user data.

Possible Solutions:

Parameterized Queries/Prepared Statements: Use parameterized queries or prepared statements to separate user input from the SQL code. This prevents the user input from being interpreted as SQL code.

Input Validation: Validate all user inputs to ensure they conform to expected formats and data types.

Least Privilege: Grant the database user account only the necessary permissions to perform the required operations.

Regular Security Audits: Conduct regular security audits and penetration testing to identify and address potential vulnerabilities.

2. Cross-Site Scripting (XSS):

What it is:

XSS vulnerabilities allow attackers to inject malicious scripts (usually JavaScript) into web pages viewed by other users. When a user visits the compromised page, the script executes in their browser, potentially leading to information theft, session hijacking, or redirection to malicious websites.

Example:

An attacker might inject a script into a comment section on a forum that steals user cookies when another user views the page.

Possible Solutions:

Output Encoding: Encode user-provided data before displaying it on the webpage. This converts potentially harmful characters into harmless HTML entities, preventing the script from executing.

Input Validation: Validate user inputs to ensure they don't contain malicious script elements.

- Content Security Policy (CSP): Implement a CSP to control the sources from which the browser can load resources, limiting the potential impact of XSS attacks.

Sanitization: Remove or neutralize any potentially harmful characters from the user input.

3. Broken Authentication:

What it is:

Broken authentication vulnerabilities occur when an application's authentication and session management mechanisms are flawed, allowing attackers to bypass login procedures or gain access to user accounts through various means.

Example:

Weak passwords, predictable session IDs, or failure to properly invalidate sessions after logout can all lead to broken authentication.

Possible Solutions:

Strong Password Policies: Enforce strong password requirements, including length and complexity.

Multi-Factor Authentication (MFA): Implement MFA to add an extra layer of security beyond just passwords.

Secure Session Management: Use strong session IDs, protect them with encryption, and ensure they are properly invalidated when users log out or when the session expires.

Regular Audits: Regularly review authentication logs for suspicious activity and implement automated alerts for failed login attempts.

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

Here's a breakdown of 5 common applications classified as either system or application software:

System Software:

1. Operating System (Windows, macOS, or Linux):

This is the core software that manages all hardware and software resources and provides basic functionalities like file management, memory management, and process management. It's essential for the computer to function.

2. device Drivers (e.g., Graphics card driver)

These drivers allow the operating system to communicate with and control specific hardware components like printers, graphics cards, and input devices.

Application Software:

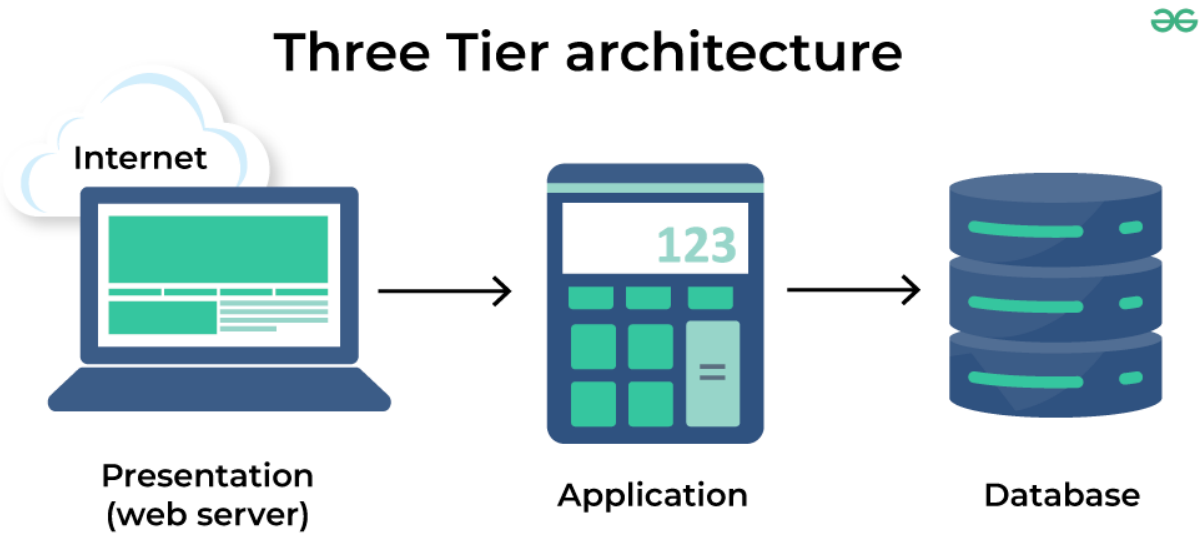
1. web browser(e.g, google chorme , fire fox): used to access and interact with websites and online content. Google chrome and firefox are not system software.

Email client(e.g, outlook, gmail): used for sending , receiving and managing emails .

3. word processor(e.g, microsoft word): used for creating, editing and formatting text documents.

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

A basic three-tier architecture for a web application consists of three layers: the Presentation Tier (user interface), the Application Tier (business logic), and the Data Tier (database). These layers interact to handl users requests and manage applicatio data



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

A three-layer architecture, commonly used in software development, separates an application into distinct layers: presentation, business logic, and data access. This structure enhances modularity, maintainability, and scalability. This case study will explore the functionality of each layer in a simplified e-commerce application

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

Software development typically involves distinct environments: development, testing, staging, and production. A virtual machine (VM) can be used to create isolated environments for each, allowing developers to work without affecting other environments.

1. Development Environment: This is where developers write, modify, and test code. It's a sandbox for experimentation and debugging. A local development environment can be set up on your computer using tools like IDEs (e.g., VS Code, Eclipse), version control (e.g., Git), and local servers (e.g., Apache, Nginx).

2. Testing Environment: This environment is used for comprehensive testing, including functional, performance, regression, and security tests. QA teams use this to identify and fix bugs before moving to staging.

3. Staging Environment: This environment is a near-replica of the production environment, used for final testing before deployment. It allows teams to catch any last-minute issues that might only appear in a production-like setting.

4. Production Environment: This is the live environment where the application is accessible to end-users.

LAB EXERCISE: Write and upload your first source code file to Github.

Write and upload your first source code file to Github.

Uploading your first source code file to GitHub

Here's how to create a new repository on GitHub and upload your first source code file:

1. Create a GitHub account

Go to GitHub and click on "Sign up".

Follow the instructions to create your account and verify your email address.

2. Install Git on your local machine

Git is the version control system that GitHub is built on.

Download and install Git from the official Git website (git-scm.com), selecting the appropriate installer for your operating system (Windows, macOS, or Linux).

Verify the installation was successful by typing git --version in your terminal or command prompt.

Configure your Git username and email using the following commands, replacing the example with your own information:

A) bash

```
B)git config --global user.name "Your Name"
  git config --global user.email
  "your_email@example.com"

C)Use code with caution.
```

3. Create a new repository on GitHub

Go to the GitHub website and log in to your account.

Click the "+" sign in the top-right corner of the navigation bar and select "New repository".

Enter a name and description for your repository.

Choose whether you want the repository to be public or private.

Select "Initialize this repository with a README".

Click "Create repository".

4. Prepare your source code file

Create or locate the source code file you want to upload on your computer.

5. Upload your file

Open your newly created repository on GitHub.

Click on the "Add file" dropdown menu, and then select "Upload files".

Drag and drop your file into the designated area or click "choose your files" to select it from your computer.

Add a commit message describing the changes you're making (e.g., "Initial commit of my first source code file").

Click "Commit changes".

Congratulations! You have successfully uploaded your first source code file to GitHub.

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

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

Create a GitHub repository and document how to commit and push code changes

This guide will walk you through the process of creating a new repository on GitHub and then cover how to make changes, commit them to your local repository, and finally push those changes to your remote GitHub repository, using the Git command line.

1. Create a new repository on GitHub

Sign in to GitHub: Go to GitHub and sign in to your account.

Navigate to the "New repository" page: In the upper-right corner of any page, select the "+" sign and then click "New repository".

Choose a repository name and description: Type a short, memorable name for your repository (e.g., "my-first-repo"). Optionally, add a description to explain the purpose of your repository.

Set visibility: Choose whether you want the repository to be public or private.

Initialize with a README (recommended): Select "Initialize this repository with a README". A README file helps describe your project. The contents of your README file are automatically shown on the front page of your repository.

Create the repository: Click "Create repository". You have successfully created your first repository on GitHub.

2. Clone the repository to your local machine

Open your repository: Navigate to the repository you just created on GitHub.

Get the clone URL: Click the green "Code" button and copy the HTTPS URL for your repository.

Open your terminal or Git Bash: On your local machine, open your preferred command-line interface.

Navigate to your desired directory: Use the cd command to change your current working directory to the location where you want to clone your repository. For example: cd Documents.

Clone the repository: Use the git clone command followed by the copied URL to clone the repository to your local machine: git clone https://github.com/your-username/your-repository-name.git.

Verify local repository: After cloning, you should see a new directory with the same name as your repository. You can navigate into it using cd your-repository-name.

3. Commit and push code changes

Make changes to your code: Open the files within your local repository in a text editor or IDE and make the necessary modifications. For example, let's edit the README.md file.

Stage your changes: Before committing, you need to stage the modified files. Use git add . to stage all changes in the current directory. Alternatively, you can stage specific files, e.g., git add README.md.

Commit the changes: Create a snapshot of the current state of your repository using the git commit command. You will need to include a concise commit message describing the changes you've made. TheServerSide suggests a subject line of 50 characters or less, written in the imperative mood (e.g., "Add new feature").

bash

git commit -m "Update README with project details"

Use code with caution.

This command takes the staged files and creates a new commit with the specified message. You can also use git commit -a -m "message" to skip the staging phase and automatically add and commit all tracked changes.

Push the changes to GitHub: To upload your local commits to your remote GitHub repository, use the git push command.

bash

git push origin main

Use code with caution.

Here, origin is the default remote name (your GitHub repository), and main is the name of your branch. If you are pushing for the first time or want to set a tracking relationship for a new branch, you might use: git push -u origin main. The -u or --set-upstream flag establishes a tracking relationship between your local and remote branches.Note: GitHub requires you to use a Personal Access Token (PAT) when pushing code from the command line using HTTPS. You can refer to this link for information on creating a PAT.

Verify changes on GitHub: Go back to your repository on GitHub. You should now see the changes you pushed reflected there, along with the commit history.

Best practices for commit messages

Be descriptive: A good commit message clearly and concisely tells what changed and why it changed. The subject line should be a short summary (under 50 characters, ideally).

Use the imperative mood: Start your commit messages with verbs in the imperative mood, such as "Fix bug," "Add new feature," or "Update documentation".

Include a body (optional): For more complex changes, provide a more detailed explanation in the body of the commit message, separated from the subject line by a blank line. The body should explain why the change was needed and the effects of the change, without going into how it was implemented.

Wrap lines: Wrap lines in the body of the commit message at 72 characters.

Small, focused commits: Aim to make small, atomic commits, where each commit addresses a single logical change. This makes it easier to review and understand changes in the future.

Use tags (optional): Some teams use tags like feat, fix, refactor, or docs to categorize commit messages.

By following these steps and best practices, you can effectively use GitHub and Git to manage your code changes.

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

Creating a GitHub repository and documenting how to commit and push code changes

This guide will walk you through the process of creating a new repository on GitHub and then explain how to commit and push code changes using Git commands.

1. Create a new repository on GitHub

Sign in to GitHub: Go to GitHub and sign in to your account.

Navigate to the "New repository" page: In the upper-right corner of any page, select the "+" sign and then click "New repository".

Choose a repository name and description: Type a short, memorable name for your repository (e.g., "my-project"). Optionally, add a description to explain its purpose.

Set visibility: Choose whether you want the repository to be public or private.

Initialize with a README (recommended): Select "Initialize this repository with a README." A README file helps describe your project, and its contents are automatically shown on the front page of your repository.

Create the repository: Click "Create repository." You have successfully created your first repository on GitHub.

2. Clone the repository to your local machine

Open your repository: Navigate to the repository you just created on GitHub.

Get the clone URL: Click the green "Code" button and copy the HTTPS URL for your repository.

Open your terminal or Git Bash: On your local machine, open your preferred command-line interface.

Navigate to your desired directory: Use the cd command to change your current working directory to the location where you want to clone your repository. For example: cd Documents.

Clone the repository: Use the git clone command followed by the copied URL to clone the repository to your local machine. For example: git clone https://github.com/your-username/my-project.git.

Verify local repository: After cloning, you should see a new directory with the same name as your repository. You can navigate into it using cd my-project.

3. Commit and push code changes

make changes to your code: Open the files within your local repository in a text editor or IDE and make the necessary modifications. For example, edit the README.md file.

Stage your changes: Before committing, you need to stage the modified files. Use git add . to stage all changes in the current directory, or use git add README.md to stage specific files.

Commit the changes: Create a snapshot of the current state of your repository using the git commit command. You will need to include a concise commit message describing the changes you've made.

bash

git commit -m "Update README with project details"

Use code with caution.

This command takes the staged files and creates a new commit with the specified message. You can also use git commit -a -m "message" to skip the staging phase and automatically add and commit all tracked changes.

Push the changes to GitHub: To upload your local commits to your remote GitHub repository, use the git push command.

bash

git push origin main

Use code with caution.

Here, origin is the default remote name (your GitHub repository), and main is the name of your branch. If you are pushing for the first time or want to set a tracking relationship for a new branch, you might use: git push -u origin main. The -u or --set-upstream flag establishes a tracking relationship between your local and remote branches.Note: GitHub requires you to use a Personal Access Token (PAT) when pushing code from the command line using HTTPS. You can refer to this link for information on creating a PAT.

Verify changes on GitHub: Go back to your repository on GitHub. You should now see the changes you pushed reflected there, along with the commit history.

Best practices for commit messages

Be descriptive: A good commit message clearly and concisely tells what changed and why it changed. The subject line should be a short summary (under 50 characters, ideally).

Use the imperative mood: Start your commit messages with verbs in the imperative mood, such as "Fix bug," "Add new feature," or "Update documentation".

Include a body (optional): For more complex changes, provide a more detailed explanation in the body of the commit message, separated from the subject line by a blank line. The body should explain why the change was needed and the effects of the change, without going into how it was implemented.

Wrap lines: Wrap lines in the body of the commit message at 72 characters.

Small, focused commits: Aim to make small, atomic commits, where each commit addresses a single logical change. This makes it easier to review and understand changes in the future.

Use tags (optional): Some teams use tags like feat, fix, refactor, or docs to categorize commit messages.

By following these steps and best practices, you can effectively use GitHub and Git to manage your code changes.

LAB EXERCISE: 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, classified as system, application, or utility, with examples:

System Software:

Operating System:

Windows, macOS, Linux (e.g., Ubuntu, Fedora), iOS, Android. These manage the computer's resources and provide a platform for other software.

Device Drivers:

Software that allows the operating system to interact with hardware devices like printers, graphics cards, etc.

Firmware:

Embedded software within hardware components like a computer's BIOS or a printer's control system.

Application Software

Web browser:

Google, chrome mozila firefox, safari, microsoft edge. They allow users to access and interact with the internet.

Office suites:

Microsoft office(word, excel, powerpoint), google workplace(docs, sheets, slides). They provide tools for word processing, spreadsheets, presentations, etc.

Multimedia players:

VLC media player, quicktime player. They allow users to play audio and video files.

Communication software:

Slack,skype,zoom,microsoft teams. They facilitate online communication and collaboration.

Graphics software: adobe photoshop, GIMP. They enable users to create, edit, and manipulate images and graphics.

Programming languages:

Python, java, c++,etc. These are used for writing and executing code.

Database software :

MySQL, posterSQL, mongoDB, etc. They allow users to store, manage, and retrieve data.

Enterprise software:

ERP system, CRM systems, etc. These are used by bussiness for various functions like resources planning and customer relationship management.

Utility software:

Disk cleanup: tools that remove unnecessary files from a hard drive to free up space.

Amtivirus software: McAfee, Norton, Windows Defender. They protect the computer from malware.

System monitoring tools: resources monitor, task manager. They allow users to track system performances and resources usage

Backup software: acronis true image, time machine (macOS). They allow users to create backups of their data.

File archivers: WinRAR, 7-Zip. They compress and decompress files.

LAB EXERCISE: Follow a GIT tutorial to practice cloning, branching, and merging repositories.

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

Okay, let's go through a step-by-step tutorial for practicing Git cloning, branching, and merging, building on the previous steps you followed to set up a repository and make your first commit

.

Git Tutorial: Cloning, branching, and merging practice

This tutorial assumes a repository (e.g., my-project) on GitHub has been created and cloned to a local machine. If not, please refer to the previous answer.

1. Cloning the repository (review)

Navigate to the directory to store project locally. For example, in a terminal: Cd Documents.

Clone the repository using the git clone command and the HTTPS URL from the GitHub repository (e.g., git clone https://github.com/your-username/my-project.git).

Change the current directory to the newly cloned repository: cd my-project.

2. Creating a new branch

Create a new branch: Use the git branch command followed by the new branch's name (e.g., feature-branch).

bash

```
A) git branch feature-branch
```

Use code with caution.

Switch to the new branch: Use the git checkout command followed by the new branch's name to move into it: git checkout feature-branch.

bash

git checkout feature-branch

Use code with caution.

Changes won't directly affect the main branch.

Verify the branch: Use git branch to list all branches and see which one is active (marked with an asterisk).

3. Making changes and committing

Modify a file: Open the README.md file (or another file in the repository) in a text editor and add new content.

Check the status: Use git status to see the changes (e.g., "modified: README.md").

Stage the changes: Use git add . to stage all modifications or git add README.md for specific files.

Commit the changes: Use git commit -m "Add new feature description" to commit the changes with a descriptive message.

Tip: Make small, frequent commits with clear messages.

4. Merging the branch

Switch back to the main branch: Use git checkout main.

Merge the feature branch into the main branch: Use git merge feature-branch.

Git attempts to merge the changes automatically. DataCamp notes that If the same lines were modified in both branches, it results in a merge conflict that needs to be resolved.

Resolve merge conflicts (if necessary):

If conflicts occur, Git stops the merge and adds special markers (like <<<<<<<, =======, >>>>>>>) to indicate the conflicting parts.

Open the file(s) with conflicts and manually edit them to combine the changes.

Once resolved, stage the file: git add <file-name>.

Commit the merge: git commit -m "Merge feature-branch into main".

Verify changes: Check the README.md file (or the file modified in the feature branch) to ensure the changes are present in the main branch.

Delete the merged branch (optional but recommended for a clean repository): Use git branch -d feature-branch.

Push the merged changes to GitHub: Use git push origin main.

Verify changes on GitHub: Go to your repository on GitHub and observe the merged changes and the commit history.

B) Following these steps allows you to clone a repository, create a new branch, make changes in that branch, commit them, merge those changes back into the main branch, and push everything to GitHub.

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

Application software encompasses a wide range of programs that enable users to perform specific tasks, significantly enhancing productivity across various domains. These programs, residing above system software, allow users to create documents, manage data, communicate, and much more. By automating tasks, streamlining workflows, and providing specialized tools, application software boosts efficiency, reduces errors, and ultimately improves overall productivity.

Types of Application Software and Their Impact on Productivity:

1. Word Processors:

These tools, like Microsoft Word or Google Docs, allow users to create, edit, and format text documents, such as reports, letters, and memos. They enhance productivity by providing features like spell check, grammar check, and formatting options, making document creation faster and more efficient.

2. Spreadsheet Software:

Applications like Microsoft Excel or Google Sheets are crucial for managing and analyzing numerical data. They offer features like calculations, charting, and data analysis, enabling users to organize information, perform complex calculations, and visualize data, leading to improved decision-making and productivity.

3. Presentation Software:

Tools like Microsoft PowerPoint or Google Slides enable users to create visually engaging presentations. They offer features like slide templates, animations, and multimedia integration, allowing users to communicate information effectively and create impactful presentations, imporoving communications and imformation dissemination.

4. Database Software:

Applications such as MySQL or Microsoft Access help manage and organize large amounts of data. They allow users to store, retrieve, and analyze data efficiently, improving data management and enabling better insights.

5. Web Browsers:

Browsers like Google Chrome or Firefox are essential for accessing information and conducting online research. They enhance productivity by providing access to a vast amount of information and facilitating online communication and collaboration.

6. Communication Software:

Applications like Slack or Microsoft Teams facilitate real-time communication and collaboration among team members. They improve productivity by enabling instant messaging, video conferencing, and file sharing, streamlining communication and reducing the need for lengthy email chains.

7. Graphics Software:

Tools like Adobe Photoshop or GIMP allow users to create and edit images and graphics. They enhance productivity by providing tools for image manipulation, design, and visual communication.

8. Specialized Software:

This category includes software tailored to specific industries or tasks, such as CAD software for engineers, or laboratory management software for scientific research. These applications are designed to automate specific tasks, improve accuracy, and enhance productivity within their respective domains.

9. Enterprise Resource Planning (ERP) Software:

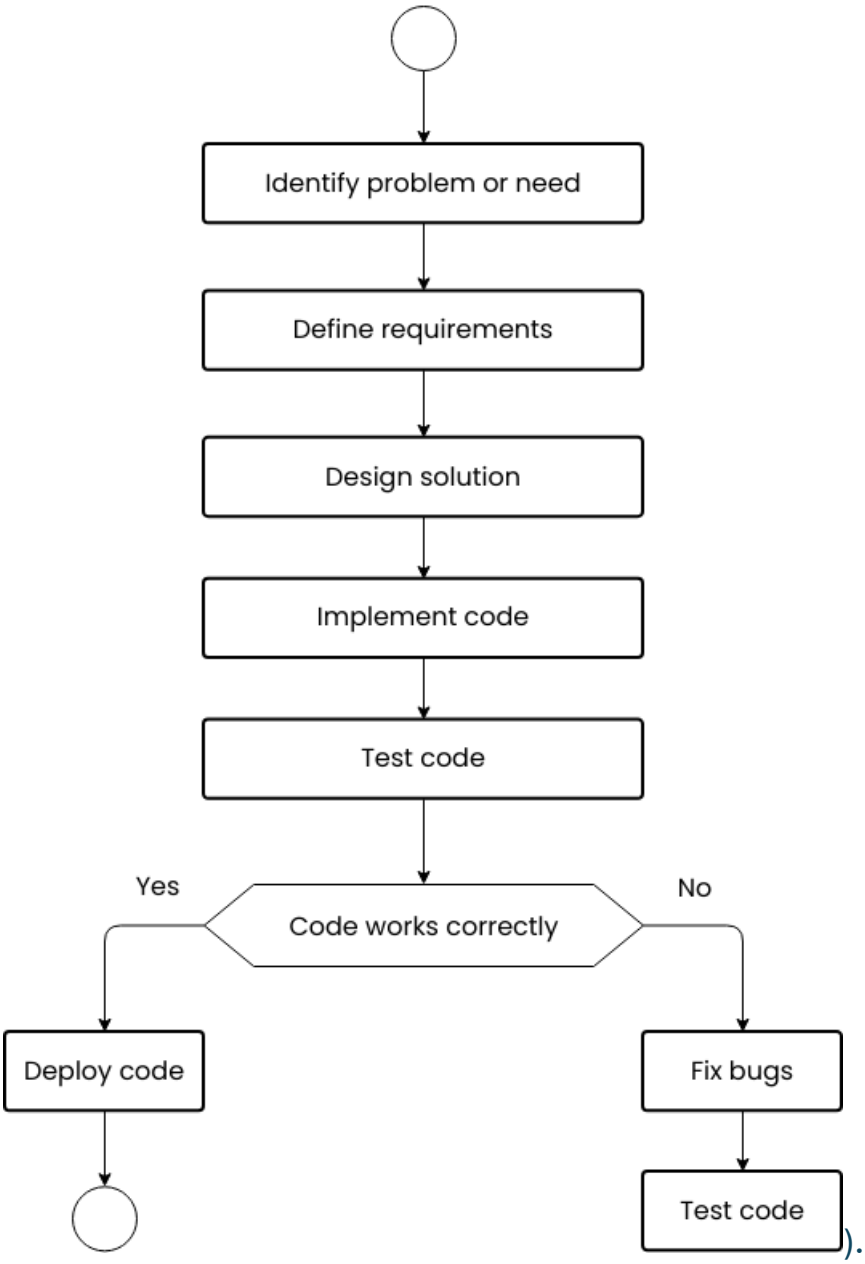
ERP systems integrate various business functions, such as finance, human resources, and supply chain management. They improve productivity by streamlining business processes, providing real-time data visibility, and enabling better decision-making.

10. Customer Relationship Management (CRM) Software:

CRM systems, like Salesforce, help businesses manage customer interactions and relationships. They improve productivity by streamlining sales processes, improving customer service, and enhancing customer retention.

In conclusion, application software plays a vital role in enhancing productivity across various domains. By automating tasks, streamlining workflows, and providing specialized tools, these applications enable individuals and organizations to achieve more in less time, ultimately boosting efficiency and effectiveness.

LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDL



).

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

Write a requirement specification for a simple library management system.

Simple library management system: requirement specification

This document outlines the requirements for a simple Library Management System (LMS), designed to automate core library operations and enhance efficiency for both staff and patrons.

1. Introduction

The LMS aims to simplify tasks like managing books, tracking loans, and maintaining user records. This will reduce manual effort, minimize errors, and improve overall library service.

2. Functional requirements

These define what the system will do.

Book Management:

Add new books to the library system, including details like title, author, ISBN, and quantity available.

Remove or update information about existing books.

Search for books by title, author, or ISBN.

Track the availability status of books.

User Management:

Register new users (patrons) with their name, ID, and contact details.

Maintain and update user information.

Provide login functionality for users.

Circulation Management:

Allow users to check out books.

Track the loan duration and due dates for borrowed books.

Enable users to return books.

Calculate and track overdue fines.

Reporting:

Generate reports on library usage, such as lists of books issued or returned within a specified period.

Generate reports on overdue books and fines.

3. Non-functional requirements

These describe how well the system performs its functions.

Usability: The system should have a user-friendly interface that is easy to navigate and understand for both librarians and patrons.

Performance: The system should respond to user requests within a reasonable time frame (e.g., search results within 2-5 seconds). It should be able to handle a moderate number of concurrent users.

Reliability: The system should be stable and operate without frequent crashes or errors. Data integrity should be maintained at all times.

Security: User accounts and personal information must be protected. Different user roles (librarian, patron) will have appropriate access rights.

Scalability: The system should be able to accommodate an increasing number of books and users as the library grows.

Maintainability: The software should be easily maintainable and allow for future updates and modifications.

Compatibility: The system should be compatible with common web browsers and operating systems.

4. Actors

The main users interacting with the system are:

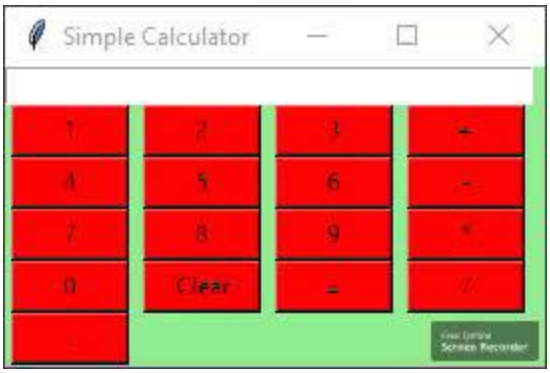
Librarian: Manages books, registers users, issues/returns books, manages fines, and generates reports.

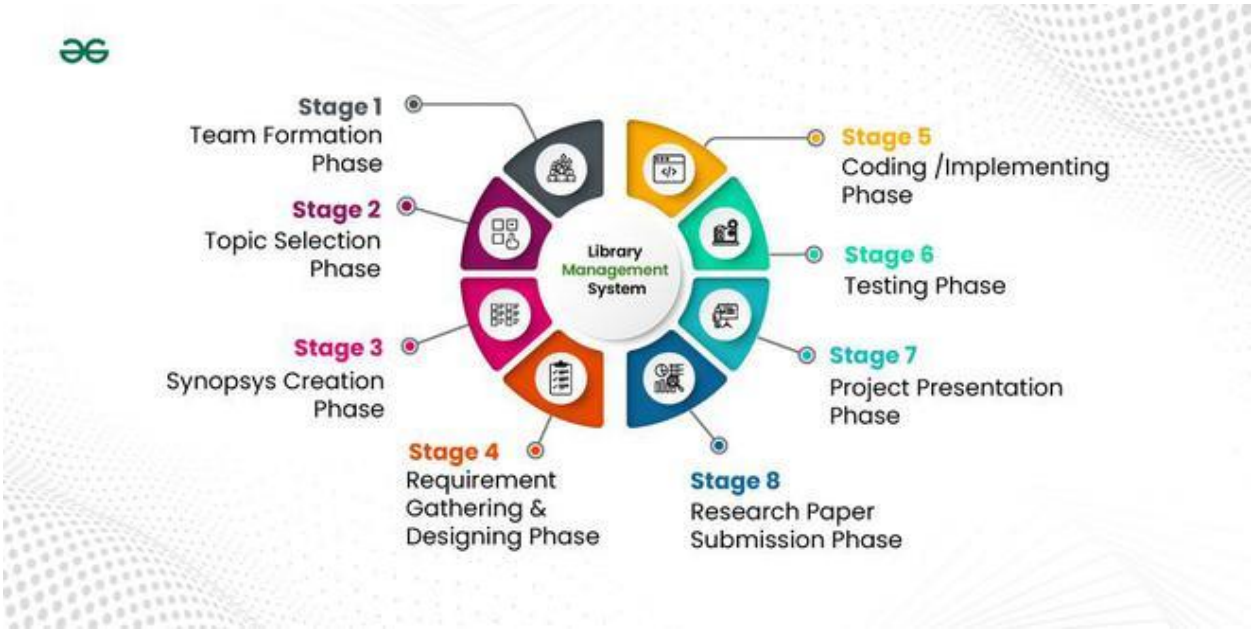
Patron/User: Searches for books, views borrowed books, checks out and returns books, and pays fines.

5. System constraints

The system will use a relational database management system (DBMS) for data storage.

The system will be accessible via a web interface.





LABEXERCISE: Perform a functional analysis for an online shopping system

Perform a functional analysis for an online shopping system

Functional analysis of an online shopping system

A functional analysis of an online shopping system involves identifying the key features and functionalities that allow users to interact with the system and fulfill their shopping needs.

Here's a breakdown of the major functional areas:

1. User account management

Registration and Login: Users can create an account, log in, and log out.

Profile Management: Users can view and update their personal information, delivery addresses, and payment methods.

Order History: Users can access their past orders, including details and tracking information.

Wishlist: Users can save products they are interested in for future purchase.

2. Product browsing and search

Product Catalog: The system should display a wide variety of products organized into categories and subcategories.

Product Search: Users can search for products using keywords, categories, or brands.

Product Filters: Users can refine search results using filters such as price, size, color, brand, etc.

Product Details: Each product should have a dedicated page with detailed descriptions, images, pricing, and availability.

3. Shopping cart and checkout

Add to Cart: Users can add desired products to a shopping cart.

Cart Management: Users can view, edit, and remove items from their cart.

Checkout Process: A streamlined checkout process, potentially with guest checkout and auto-fill options.

Payment Gateway Integration: Secure processing of various payment methods like credit cards, debit cards, digital wallets, etc.

Shipping Options: Offer different shipping methods (standard, expedited) with real-time rate calculations.

4. Order management and fulfillment

Order Creation: The system generates orders after successful checkout.

Order Processing: The fulfillment team receives and processes orders, including picking, packing, and generating shipping labels.

Order Tracking: Users and administrators can track the real-time status of orders, from processing to delivery.

Returns Processing: Handle returns and exchanges according to the defined return policy.

5. Security and compliance

Secure Payment Processing: PCI DSS compliance, SSL certificates, and secure payment gateways are crucial for protecting sensitive payment information.

User Data Protection: Implement security measures like multi-factor authentication (MFA) and data encryption to safeguard user data.

Fraud Detection and Prevention: Incorporate tools and systems to detect and prevent fraudulent transactions.

6. Admin functions

Inventory Management: Track product stock levels and update availability in real-time.

Order Management: Manage and update order statuses, view customer information, and handle order-related queries.

Content Management (CMS): Manage product descriptions, images, promotional banners, and other website content.

Reporting and Analytics: Generate reports on sales, revenue, customer behavior, traffic sources, and other key metrics to inform business decisions.

Promotions and Discounts: Create and manage coupon codes, loyalty programs, and other promotional offers.

This functional analysis provides a comprehensive overview of the essential functionalities of an online shopping system. The specific implementation of these features can vary based on the platform, business model, and target audience.

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

Design a basic system architecture for a food delivery app.

Basic system architecture for a food delivery app

A food delivery app requires a well-structured architecture to ensure seamless functionality, real-time tracking, and a positive user experience. The system needs to effectively manage interactions between customers, restaurants, and delivery partner Restaurant Dashboard: s.

Here's a breakdown of the core components for a basic system architecture:

1. User interfaces (front-end)

Customer App: This interface allows customers to browse restaurants and menus, place orders, make payments, and track deliveries.

Enables restaurants to manage orders, update menus and pricing, track delivery partners, and view payment information.

Delivery Partner App: Provides delivery partners with information on delivery requests, customer details, navigation assistance, and the ability to update order statuses.

Admin Dashboard: A centralized platform for the platform administrators to manage users, restaurants, delivery partners, and analytics.

2. Backend application layer

The backend acts as the central nervous system of the app, handling the logic, data, and communication between different components. It typically includes:

User Service: Manages user registration, authentication, and profiles.

Restaurant Service: Handles restaurant data, including menus, ratings, and locations.

Order Service: Manages the entire order lifecycle, from placement to delivery completion.

Delivery Service: Assigns delivery partners to orders, manages their availability, and tracks their location.

Payment Service: Processes payments securely through integrated payment gateways.

Notification Service: Sends notifications and updates to customers, restaurants, and delivery partners via push notifications, SMS, or email.

3. Database layer

The database layer is crucial for storing and retrieving essential data quickly and efficiently. This includes data on customers, restaurants, orders, and drivers. Relational databases like PostgreSQL or MySQL are suitable for structured data, while NoSQL databases such as MongoDB or Firebase offer flexibility for evolving data.

4. Cloud infrastructure and API integrations

Cloud Hosting Platforms: Platforms like AWS, Google Cloud, or Microsoft Azure provide scalable resources.

Payment Gateway APIs: Integration with services like Stripe or PayPal for secure transactions.

Mapping and Geolocation APIs: Services such as Google Maps Platform or Mapbox are used for tracking and location features.

Push Notification Services: Services like Firebase Cloud Messaging (FCM) or Apple Push Notification Service (APNs) enable real-time notifications.

SMS Services: APIs like Twilio or Nexmo are used for automated updates.

5. Caching and message queues

Caching: Mechanisms such as Redis or Memcached can be used to improve performance by storing frequently accessed data.

Message Queues: Brokers like Kafka or RabbitMQ facilitate the asynchronous processing of tasks, enhancing scalability.

This architecture forms a basis for a food delivery application, using a microservices approach for modularity and independent scaling of components.

Design a basic system architecture for a food delivery app.

Food delivery app: system architecture design

A food delivery app requires a robust and scalable architecture to handle the complex interactions between customers, restaurants, and delivery personnel. A typical architecture follows a microservices approach, dividing the system into independent, interconnected services.

Here's a breakdown of the essential components and their interactions:

1. User interfaces (front-end layer)

The front-end layer consists of several user interfaces tailored for different roles within the system:

Customer App: For browsing, ordering, payment, and tracking.

Restaurant Dashboard: For managing orders, menus, and finances.

Delivery Partner App: For receiving requests, navigation, and updating statuses.

Admin Dashboard: A central portal for system oversight and management.

2. Backend services (application layer)

The backend is built with microservices to manage various functionalities:

User Service: Manages user accounts and profiles.

Restaurant Service: Handles restaurant data.

Order Service: Manages the ordering process.

Delivery Service: Manages delivery logistics and assignments.

Payment Service: Processes payments securely.

Notification Service: Sends real-time updates.

Other Services: May include features like promotions, reviews, and recommendations.

3. Database layer

This layer is responsible for storing and managing all essential data, including user, restaurant, and order information. Relational databases like PostgreSQL or MySQL and NoSQL databases like MongoDB or Firebase are common choices.

4. Cloud infrastructure & API integrations

The system leverages cloud platforms such as AWS, Google Cloud, or Microsoft Azure for scalability. API integrations are crucial for connecting with external services like Google Maps for tracking, various payment gateways for transactions, SMS/email APIs for notifications, and AI engines for recommendations.

5. System interactions (how it works)

The process typically flows as follows: A customer places an order via the app; the restaurant is notified and accepts the order; a driver is assigned; real-time tracking begins; the driver delivers the order and payment is processed; finally, the customer can leave reviews.

Key design principles

Important design principles include:

Performance: Fast response times and low latency.

Scalability: Ability to handle increasing users and orders.

Security: Protecting sensitive user and payment data.

Reliability: System robustness against failures.

User-friendliness: Intuitive and easy-to-use interfaces.

By implementing this architecture, a food delivery app can efficiently manage the entire delivery process, providing a good user experience.

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

Develop test cases for a simple calculator program.

A test case for a simple calculator program aims to verify that the calculator correctly performs basic arithmetic operations (addition, subtraction, multiplication, division) and handles edge cases appropriately

.

Here's a breakdown of test cases, categorized for clarity:

Functional test cases

These test cases verify the core functionality of the calculator.

Arithmetic Operations:

Addition of two positive integers (e.g., 5 + 3 = 8).

Addition of two negative integers (e.g., -5 + (-3) = -8).

Addition of a positive and a negative integer (e.g., 5 + (-3) = 2).

Subtraction of two positive integers (e.g., 8 - 3 = 5).

Subtraction of two negative integers (e.g., -8 - (-3) = -5).

Subtraction of a negative and a positive integer (e.g., -5 - 3 = -8).

Multiplication of two positive integers (e.g., 5 * 3 = 15).

Multiplication of two negative integers (e.g., -5 * (-3) = 15).

Multiplication of a negative and a positive integer (e.g., -5 * 3 = -15).

Division of two positive integers (e.g., 10 / 2 = 5).

Division of two negative integers (e.g., -10 / -2 = 5).

Division of a positive and a negative integer (e.g., 10 / -2 = -5).

Division of zero by a number (e.g., 0 / 5 = 0).

Order of Operations (BODMAS/BIDMAS):

Verify calculations involving parentheses and different operators (e.g., (2 + 3) * 4 = 20).

Decimal Numbers:

Perform operations with decimal numbers (e.g., 2.5 + 1.2 = 3.7).

Clear and Delete Functions:

Verify the "C" or clear button clears the display and resets the calculation.

Verify the backspace or delete button deletes the last entered digit or operator.

Equality Button:

Verify that the equals button ("=") computes and displays the final result accurately.

Edge case test cases

These cases test the boundaries and potential problematic scenarios of the calculator.

Division by Zero:

Attempt to divide any number by zero and verify that the calculator displays an error message (e.g., "Error" or "Cannot divide by zero").

Large Numbers:

Perform calculations with very large numbers to check for overflow issues or handling of exponential notation.

Negative Numbers:

Perform operations involving large negative numbers.

Zero and Small Decimal Values:

Perform calculations with numbers near zero (e.g., 0.0000001 / 0.0000002) to check for precision and handling of small decimals.

User interface (UI) test cases

These cases focus on the visual elements and user interaction with the calculator.

Button Functionality:

Verify all numerical buttons (0-9) and operator buttons (+, -, *, /) function correctly when clicked.

Verify other functional buttons (e.g., decimal point, equals, clear) function as expected.

Display:

Verify that the calculator display correctly shows input numbers and calculated results.

Check for readability of text and numbers on the display.

Check how long numbers are displayed (e.g., exceeding the display limit might require exponential form).

Layout and Responsiveness:

Check that the buttons are positioned correctly and are not too close together.

Verify the calculator's appearance and functionality across different screen resolutions or window sizes (for software-based calculators).

Error Messages:

Verify error messages are displayed clearly and correctly positioned (e.g., for division by zero).

Input validation test cases

These test cases ensure the calculator handles invalid inputs gracefully.

Invalid Characters:

Attempt to input non-numeric characters or symbols (e.g., "12a+34", "10$5") and verify appropriate error messages are displayed.

Empty Input:

Try to perform operations with empty input and check for error messages.

Multiple Operators:

Input multiple consecutive operators (e.g., "5++3") and verify the behavior (e.g., the last operator overrides the previous one, or an error is displayed).

Mismatched Parentheses:

Test expressions with mismatched parentheses (e.g., "(5 + 3") and check for error messages.

By systematically testing these various scenarios, you can ensure the robustness and accuracy of your simple calculator program.

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

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

Here's a real-world case where a software application required critical maintenance, highlighting the importance of regular updates and bug fixes:

The Knight Capital Group Trading Error (2012)

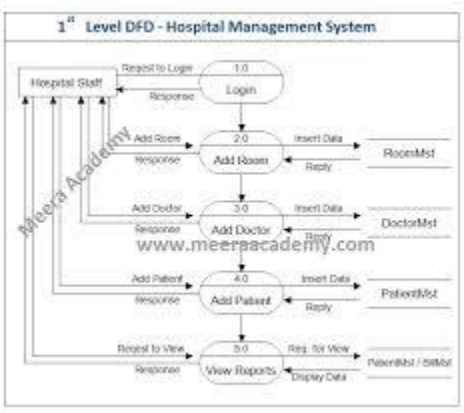
The Challenge: In 2012, Knight Capital Group, a global financial services firm, experienced a catastrophic software malfunction during a major system deployment to the stock market.

The Incident: A newly deployed auto mated trading algorithm had a glitch that caused it to rapidly buy and sell millions of shares in over a hundred stocks.

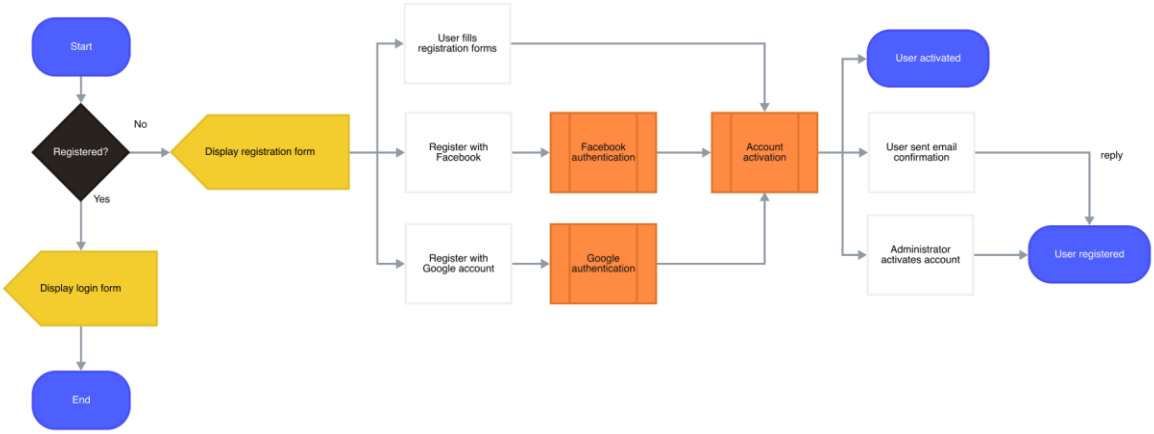
The Consequence: Within a mere 45 minutes, this faulty software deployment led to losses exceeding $440 million for Knight Capital Group, according to devlo - AI. This incident caused significant disruption in the market and highlighted the vital need for thorough testing and careful deployment of software updates in critical systems like financial trading platforms.

The Underlying Issue: The core issue stemmed from a lack of adequate testing of the updated software and insufficient safeguards in place during the deployment process. This case emphasizes that software maintenance isn't just about fixing existing bugs, but also about proactively preventing new ones through rigorous testing, especially in critical applications where even small errors can have devastating consequences.

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



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


Start
Registered?
No
Yes
Display registration form
Display login form
End
User fills registration forms
Register with Facebook
Register with Google account
Facebook authentication
Google authentication
Account activation
User activated
User sent email confirmation
Administrator activates account
reply
User registered