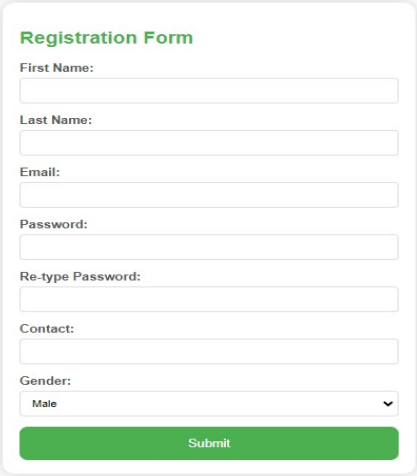
FULL STACK DEVELOPMENT

Unit-1

Imp Q & A

1) Write a simple program by using html tags (any program example html registration or login page)

Ans) HTML Registration Form is a web-based form designed to collect user information such as name, email, password, and other details. It typically includes input fields, validation, and a submit button to register users on a website.



HTML Registration Form

How to Create HTML Registration Form

- **Basic Structure:** A responsive registration form is created using HTML and CSS for styling, with a simple layout and input fields.
- **Form Inputs:** The form includes inputs for first name, last name, email, password (with validation pattern), contact number, and gender selection.
- **Styling:** CSS is used for a modern, centered form layout with a clean design, including background color, box-shadow, and input styling.
- **Password Validation:** A password pattern enforces strong security, requiring at least one number, letter, symbol, and a minimum length of 8 characters.
- **Submit Button:** A styled submit button is provided with a cursor: pointer for user interaction.

Example: Implementation of a Form with multiple input tags.

<!DOCTYPE html>

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registration Form</title>
</head>
<body>
  <h2>Registration Form</h2>
  <form>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required><br><br>
    <button type="submit">Register</button>
  </form>
</body>
</html>
```

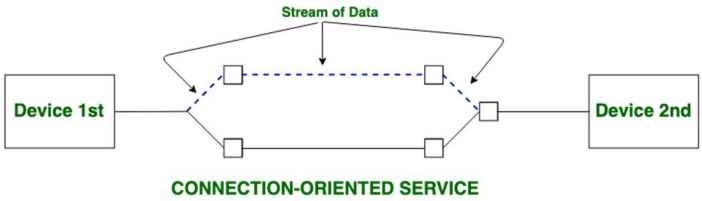
2) What is connection between client and server connection oriented or connection less explain it?

Ans) What is a Connection-Oriented Service?

Connection-oriented services involve setting up a dedicated path between the source and destination before data transfer begins. These services ensure that data is delivered in the correct sequence and without errors. In a connection-oriented service, the Handshake method is used to establish the connection between sender and receiver. Before data transmission starts, connection-oriented services create a dedicated communication channel between the sender and the recipient. As the connection is kept open until all data is successfully transferred, this guarantees dependable data delivery. One example is TCP (Transmission Control Protocol), which ensures error-free and accurate data packet delivery.

Examples of Connection-Oriented Services

- TCP (Transmission Control Protocol) in the TCP/IP suite.
- Telephone calls in traditional telecommunication systems.



Key Features of Connection-Oriented Services

- **Dedicated Connection** : A logical or physical connection is established before data transfer.
- **Reliable Transmission** : Data is transmitted with error checking, acknowledgments, and retransmissions in case of errors.
- **Sequencing** : Data packets arrive at the destination in the correct order.
- **Higher Overhead** : Establishing and maintaining a connection involves additional overhead.

Advantages of Connection-Oriented Services

- **Reliable Data Transfer** : Ensures that all data reaches its destination without errors.
- **Data Sequencing** : Packets are delivered in the correct order.
- **Error Correction** : Mechanisms are in place to detect and correct errors during transmission.
- **Guaranteed Delivery** : Retransmissions occur if data is lost.

Disadvantages of Connection-Oriented Services

- **Higher Latency** : Establishing a connection adds latency before data transfer begins.
- **More Overhead** : Requires more resources for maintaining the connection, acknowledgments, and retransmissions.
- **Less Efficient for Small Transfers** : For short messages, the overhead of connection setup can outweigh the benefits.

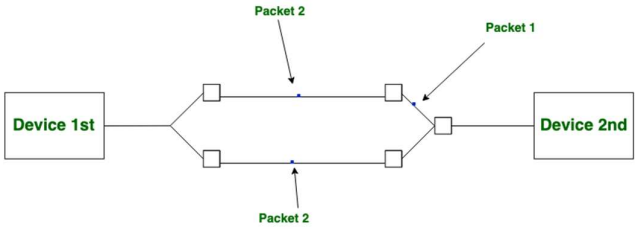
What is Connection-Less Service?

Connectionless services send data without establishing a dedicated connection between the source and destination. Each data packet is treated independently, and there is no guarantee of delivery or sequencing. Connection-less Service does not give a guarantee of reliability. In this, Packets do not follow the same path to reach their destination. Connectionless Services deliver individual data packets without first making a connection. Since each packet is sent

separately, delivery, order, and mistake correction cannot be guaranteed. As a result, the service is quicker but less dependable. UDP (User Datagram Protocol) is one example, which is frequently used for streaming where dependability is not as important as speed.

Examples of Connectionless Services

- UDP (User Datagram Protocol) in the TCP/IP suite.
- Postal services (analogous to sending letters without confirmation of receipt).



CONNECTIONIESS SERVICE

Key Features of Connectionless Services

- **No Connection Setup** : Data is sent directly without establishing a prior connection.
- **Independent Packets** : Each packet is treated individually and may take different routes to the destination.
- **Faster Transmission** : No time is spent establishing or tearing down a connection.
- **Unreliable** : No acknowledgment, retransmission, or error correction is performed.

Advantages of Connectionless Services

- **Low Latency** : Data is transmitted immediately without waiting for a connection to be established.
- **Efficient for Small Transfers** : Ideal for small, time-sensitive messages like DNS lookups or VoIP.
- **Scalable** : Suitable for systems with many simultaneous users, as no connection needs to be maintained.

Disadvantages of Connectionless Services

- **Unreliable** : Data packets may be lost, duplicated, or arrive out of order.
- **No Error Handling** : No built-in mechanisms for retransmissions or error correction.
- **Unsuitable for Large Transfers** : Not ideal for applications requiring reliable and ordered delivery.

Difference Between Connection-oriented and Connectionless Services

Connection-oriented services, like TCP, ensure reliable data transfer, while connection-less services, like UDP, offer faster, less secure communication.

Connection-oriented Service	Connection-less Service
Connection-oriented service is related to the telephone system.	Connection-less service is related to the postal system.
Connection-oriented service is preferred by long and steady communication.	Connection-less Service is preferred by bursty communication.
Connection-oriented Service is necessary.	Connection-less Service is not compulsory.
Connection-oriented Service is feasible.	Connection-less Service is not feasible.
In connection-oriented Service, Congestion is not possible.	In connection-less Service, Congestion is possible.
Connection-oriented Service gives the guarantee of reliability.	Connection-less Service does not give a guarantee of reliability.
Includes error detection, correction, and retransmission.	No error handling; errors are not corrected.
In connection-oriented Service, Packets follow the same route.	In connection-less Service, Packets do not follow the same route.
Ensures data is delivered in the correct order.	Data may arrive out of order or not at all.
Less scalable due to the need for maintaining connections.	Highly scalable for large networks with many users.

Connection-oriented Service	Connection-less Service
Higher overhead due to connection setup and maintenance.	Lower overhead as no connection is required.
Connection-oriented services require a bandwidth of a high range.	Connection-less Service requires a bandwidth of low range.
Ex: TCP (Transmission Control Protocol)	Ex: UDP (User Datagram Protocol)
Connection-oriented requires authentication.	Connection-less Service does not require authentication.

3) What is Git and GitHub?

Ans) Git: Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

GitHub: GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Below is a table of differences between Git and GitHub:

S.No.	Git	GitHub
1.	Git is a software.	GitHub is a service.
2.	Git is a command-line tool	GitHub is a graphical user interface
3.	Git is installed locally on the system	GitHub is hosted on the web

S.No.	Git	GitHub
4.	Git is maintained by linux.	GitHub is maintained by Microsoft.
5.	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6.	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7.	Git was first released in 2005.	GitHub was launched in 2008.
8.	Git has no user management feature.	GitHub has a built-in user management feature.
9.	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10.	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11.	Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
12.	Git competes with CVS, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, Azure DevOps Server, etc.

4) Explain about Git Version Control briefly Git commit and staging tags?

Ans) Git is a version control system that's widely used for tracking changes in source code during software development. Here are some key points about Git:

What is Git?

- **Distributed Version Control System (DVCS)**: Git allows multiple developers to work on a project concurrently without interfering with one another. Each developer has a full copy of the entire repository.
- **Snapshots**: Instead of tracking changes to files directly, Git takes snapshots of the project at each commit, making it easy to navigate the project's history.
- **Branches**: Developers can create branches to work on different features or experiments independently. Once the feature is ready, it can be merged back into the main branch.

Why Use Git?

- **Collaboration**: Git facilitates collaboration by allowing multiple developers to work on the same project simultaneously.
- **History Tracking**: Keeps a detailed history of changes, enabling developers to revert to previous states if needed.
- **Branching and Merging**: Supports non-linear development through branches, which can be merged back into the main codebase.
- **Distributed Nature**: Every developer has the full repository, making it possible to work offline and merge changes later.

Basic Git Commands

- git init: Initialize a new Git repository.
- git clone [url]: Clone an existing repository from a remote location.
- git status: Check the status of your files in the working directory and staging area.
- git add [file]: Add files to the staging area to prepare for a commit.
- git commit -m "message": Commit changes to the repository with a descriptive message.
- git push: Push local changes to a remote repository.
- git pull: Fetch and integrate changes from a remote repository.
- git branch: List all branches in the repository or create a new branch.
- git checkout [branch]: Switch to a different branch.
- git merge [branch]: Merge changes from one branch to another.

Example Workflow

1. Clone the repository: git clone [url]
2. Create a new branch: git branch new-feature
3. Switch to the new branch: git checkout new-feature
4. Make changes and add them: git add .
5. Commit the changes: git commit -m "Add new feature"
6. Push the changes: git push origin new-feature
7. Merge the changes back to the main branch when ready: git checkout main and git merge new-feature

Benefits of Git

- **Efficiency**: Both in terms of performance and space since each snapshot is stored efficiently.
- **Flexibility**: Supports various workflows and development practices.
- **Community Support**: With Git being widely adopted, there is vast community and tooling support for it.

Git is a powerful tool that can streamline your development process, improve collaboration, and help maintain a clear historical record of changes.