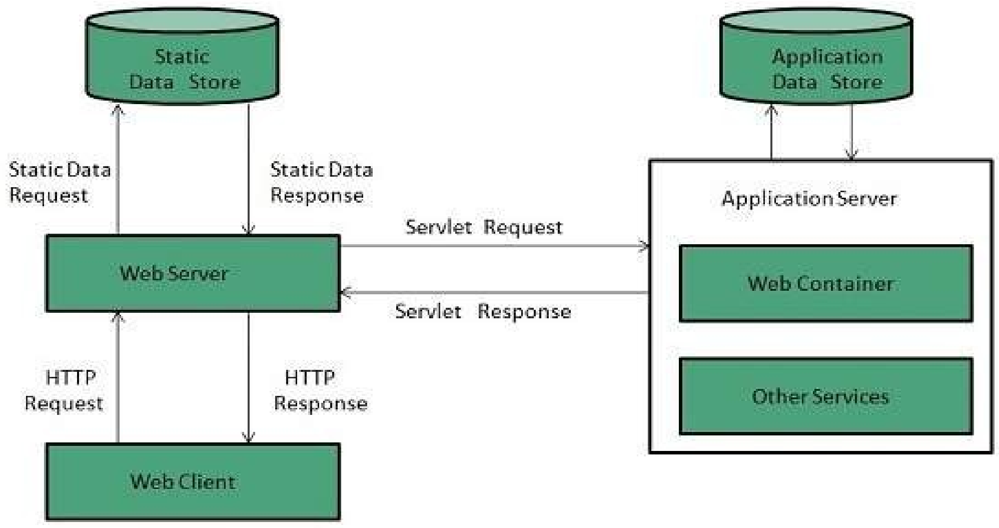
4. The Server Tier

Web Server Concepts

Web Server

A web server is a combination of software application and hardware device that serves content to users over the internet. It processes incoming network requests over the Hypertext Transfer Protocol (HTTP) and delivers web pages, files, or other resources to clients, such as web browsers.



When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response. If the requested web page is not found, web server will then send an HTTP response: **Error 404 Not found**. If client has requested for some other resources, then the web server will contact to the application server and data store to construct the HTTP response.

Here are the key components and functions of a web server:

1. **HTTP Protocol Handling**:Web servers primarily use the HTTP protocol to communicate with clients. HTTP is a stateless protocol, meaning each request from a client is independent, and the server doesn't retain information about previous requests.
2. **Request Handling:**
   - When a user makes a request by entering a URL in a web browser or clicking a link, the browser sends an HTTP request to the web server.
   - The web server processes this request, interprets the URL, and determines the appropriate action to take.
3. **Resource Retrieval:**
   - Web servers retrieve requested resources such as HTML files, images, stylesheets, JavaScript files, etc., from the file system or other sources.
   - Dynamic content may be generated on-the-fly by interacting with a database or other server-side technologies.
4. **Content-Type Negotiation:** Web servers determine the type of content being

requested based on the file extension or other metadata. Common content types include HTML, CSS, JavaScript, images, and more.

5. **Response Generation:**
   - After processing the request and retrieving the necessary resources, the web server generates an HTTP response.
   - The response includes a status code (indicating success, redirection, or an error) and the requested content.
6. **Connection Handling:** Web servers manage multiple connections simultaneously. They use mechanisms like connection pooling to efficiently handle incoming requests without creating a new process or thread for each one.
7. **Security**:
   - Web servers often include security features to protect against common threats, such as denial-of-service attacks, cross-site scripting (XSS), and SQL injection.
   - Secure Sockets Layer (SSL) or its successor, Transport Layer Security (TLS), may be used to encrypt data during transmission, ensuring secure communication between the client and server.
8. **Logging and Monitoring**: Web servers log information about incoming requests, errors, and other relevant data. These logs are useful for troubleshooting, performance monitoring, and security analysis.
9. **Configuration**: Web servers are configurable to suit the specific needs of the applications they serve. Configuration settings include options for performance tuning, security, and defining how requests are handled.
10. **Examples of Web Servers**:
    a) **Apache HTTP Server**: Developed by Apache Software Foundation, it is a free and open source web server for Windows, Mac OS X, Unix, Linux, Solaris and other operating systems; it needs the Apache license.
    b) **Nginx**: A popular open source web server for administrators because of its light resource utilization and scalability. It can handle many concurrent sessions due to its event-driven architecture. It can also be used as a proxy server and load balancer.
    c) **Microsoft Internet Information Services (IIS):** Web server developed by Microsoft for Windows Server. It is not open sourced, but widely used.
    d) **Lighttpd :** A free web server that comes with the FreeBSD operating system. It is seen as fast and secure, while consuming less CPU power.
    e) **Sun Java System Web Server:** A free web server from Sun Microsystems that can run on Windows, Linux and Unix. It is well-equipped to handle medium to large websites.

In summary, a web server plays a crucial role in the delivery of web content by handling client requests, retrieving resources, generating responses, and ensuring the security and efficiency of the communication process. Different web servers may have varying features and configurations, but they all serve the fundamental purpose of facilitating the exchange of data between clients and servers on the internet.

Short Overview of basic Protocols

Network protocols are a set of rules outlining how connected devices communicate across a network to exchange information easily and safely. They constitute a set of rules for routing and addressing packets of data, allowing them to travel across networks and reach their correct destinations. Protocols serve as a common language for devices, enabling communication regardless of differences in software, hardware, or internal processes.

The Internet Protocol (IP) is the method or protocol by which data is sent from one computer to another on the internet. Each computer (known as a host) on the internet has at least one IP address that uniquely identifies it from all other computers on the internet.

Data traversing the Internet is divided into smaller pieces called packets. IP information is attached to each packet, helping routers send packets to the correct destination. Every device or domain connecting to the Internet is assigned an IP address, and as packets are directed to the IP address attached to them, data arrives where it is needed.

Once the packets reach their destination, they are handled differently depending on the transport protocol used in combination with IP. The most common transport protocols are TCP and UDP.

IP is the defining set of protocols that enable the modern internet. It was initially defined in May 1974 in a paper titled "A Protocol for Packet Network Intercommunication," published by the Institute of Electrical and Electronics Engineers and authored by Vinton Cerf and Robert Kahn.

Some protocols used in internet are described below

1. **Transmission Control Protocol (TCP):** TCP, or Transmission Control Protocol, is one of the core protocols of the Internet Protocol (IP) suite, providing reliable, connection-oriented communication between devices on a network. TCP ensures the successful and ordered delivery of data packets between applications running on devices across a network. It breaks data into packets, sends them, and then ensures they are received correctly.
2. **Internet Protocol (IP):** The IP protocol, or Internet Protocol, is a fundamental communication protocol that facilitates the transmission of data between devices on a network. It is a key component of the Internet Protocol Suite, commonly referred to as TCP/IP. The IP protocol provides a standardized set of rules for addressing and routing data packets across networks.
3. **Hypertext Transfer Protocol (HTTP)**: HTTP, or Hypertext Transfer Protocol, is the foundation of data communication on the World Wide Web. It is an application layer protocol that facilitates the transfer of information between a client (such as a web browser) and a server. HTTP is a stateless protocol, meaning that each request from a client to a server is independent and doesn't retain any information about previous requests. HTTP typically uses port 80 for communication
4. **Hypertext Transfer Protocol Secure (HTTPS)**: HTTPS is a secure version of HTTP. It adds a layer of encryption through SSL/TLS protocols to ensure the confidentiality and integrity of data during transmission. HTTPS URLs begin with "https://" and uses port 443 for communication.

   Security Features:

   - Encryption: HTTPS encrypts the data being transmitted, making it more difficult for unauthorized parties to intercept and decipher.
   - Data Integrity: It ensures that the data remains unchanged during transmission, preventing tampering or modification by attackers.

- Authentication: HTTPS provides a way for the user's browser to verify the identity of the website's server, helping to prevent man-in-the-middle attacks.

5. **File Transfer Protocol (FTP)**: FTP stands for File Transfer Protocol, and it is a standard network protocol used to transfer files from one host to another over a TCP-based network, such as the internet. It is commonly used for uploading and downloading files between a client and a server.
6. **Simple Mail Transfer Protocol (SMTP):** SMTP stands for Simple Mail Transfer Protocol. It is a communication protocol widely used for electronic mail (email) transmission over the internet. SMTP is the set of rules governing the interaction between the mail servers to send and receive emails.
7. **User Datagram Protocol** : UDP, or User Datagram Protocol, is a connectionless transport protocol that operates at the transport layer of the Internet Protocol (IP) suite. Unlike TCP (Transmission Control Protocol), UDP does not establish a reliable, connection-oriented communication channel before data transfer. Instead, it provides a lightweight, best-effort delivery mechanism for sending data between devices on a network.
8. **Post Office Protocol (POP)** and **Internet Message Access Protocol (IMAP):** These protocols are used by email clients to retrieve messages from a mail server. POP is known for downloading emails to the local device, while IMAP allows users to manage emails directly on the server.
9. **Domain Name System (DNS)**: Domain Name System (DNS) is a protocol that allows us to use human readable names to communicate over networks, rather than having to manage and memorize IP addresses. DNS translates human-readable domain names into IP addresses. It is crucial for locating resources on the Internet by associating domain names with numeric IP addresses.
10. **Dynamic Host Configuration Protocol (DHCP)**: DHCP is used to automatically assign IP addresses and other network configuration information to devices on a network.
11. **Border Gateway Protocol (BGP)**: BGP is a routing protocol used to exchange routing and reachability information between different autonomous systems on the Internet.
12. **WebSocket**: WebSocket is a communication protocol that provides full-duplex communication channels over a single TCP connection. It is commonly used in web applications for real-time communication.

HTTP request and response

Introduction

HTTP stands for hypertext transfer protocol. Using this protocol the client sends a request to the server and based on the request the server and the web browser respond to the client.

In short it's a method where one computer (the client) communicates with another (the server).

Making HTTP Request

Once an HTTP connection is established between the client and server, the client sends a request often in the form of binary data, asking the server to provide specific files or information. This HTTP request is a message from the client, typically a web browser or application, to the server, requesting a particular resource or action. The request follows a defined format.

What's actually inside the request

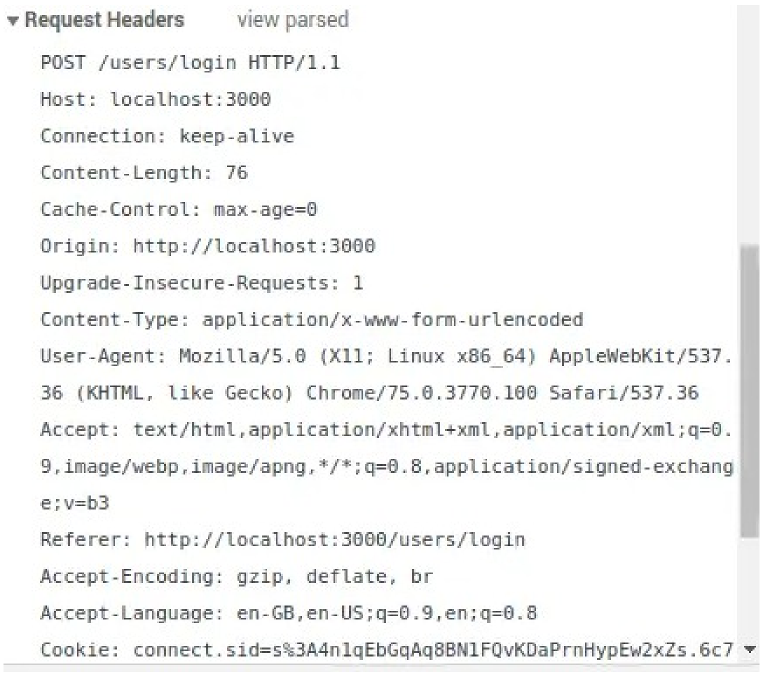
Every HTTP request contains three elements which are:- Request Line, Request Header, Body of Request(optional).

Request line:-

- It specifies the method, which tells the server what to do with the information or resource.
- It contains the URL of the request which is used to find the resource on the server.
- It also specifies HTTP protocol version being used (Ex. HTTP/ 1.0 or HTTP/1.1)

Request Header:- It consists of 0 or more headers.
The headers are used to pass more information about the request so that using the request headers the server knows how to deal with the information the client is demanding.

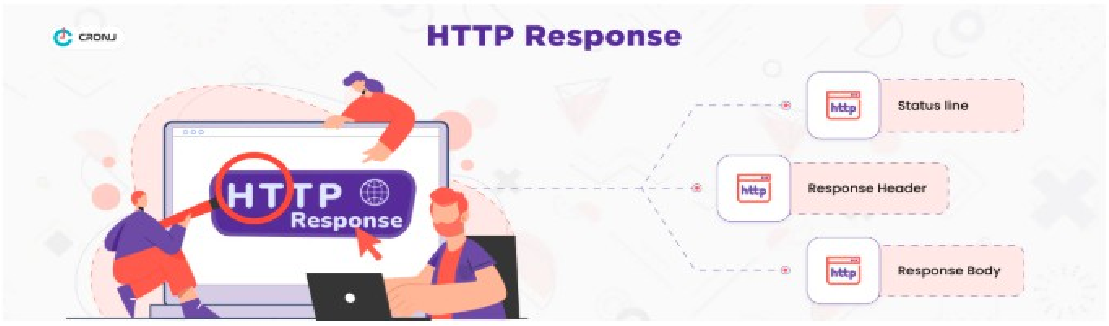
e.g. User-Agent: Mozilla/5.0
Accept: text/html



```
▾ Request Headers      view parsed
  POST /users/login HTTP/1.1
  Host: localhost:3000
  Connection: keep-alive
  Content-Length: 76
  Cache-Control: max-age=0
  Origin: http://localhost:3000
  Upgrade-Insecure-Requests: 1
  Content-Type: application/x-www-form-urlencoded
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.
  36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36
  Accept: text/html,application/xhtml+xml,application/xml;q=0.
  9,image/webp,image/apng,*/*;q=0.8,application/signed-exchang
  e;v=b3
  Referer: http://localhost:3000/users/login
  Accept-Encoding: gzip, deflate, br
  Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
  Cookie: connect.sid=s%3A4n1qEbGqAq8BN1FQvKDaPrnHypEw2xZs.6c7
```

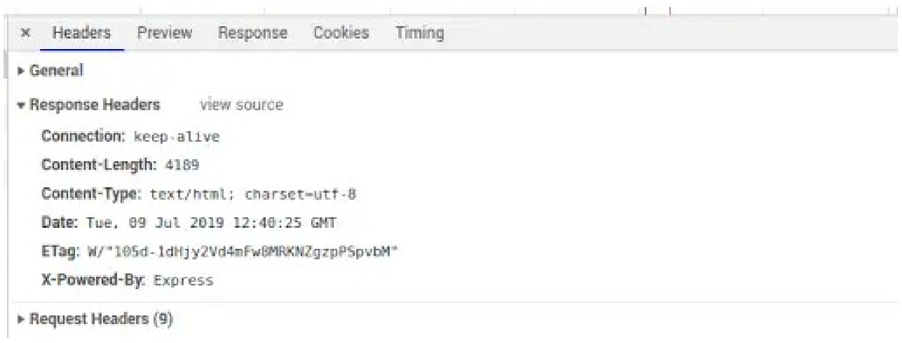
Request Body:-
This is an optional part of the HTTP request which is used to send additional data to the server.
Let's consider a post request that we make to submit a form data. The data from the form should be present in the request body of the HTTP request so that the server can access the data and save it for future use.

HTTP Response
An HTTP response is the message sent by the server back to the client in response to the HTTP request. It includes Status line, Response Header and Response Body.



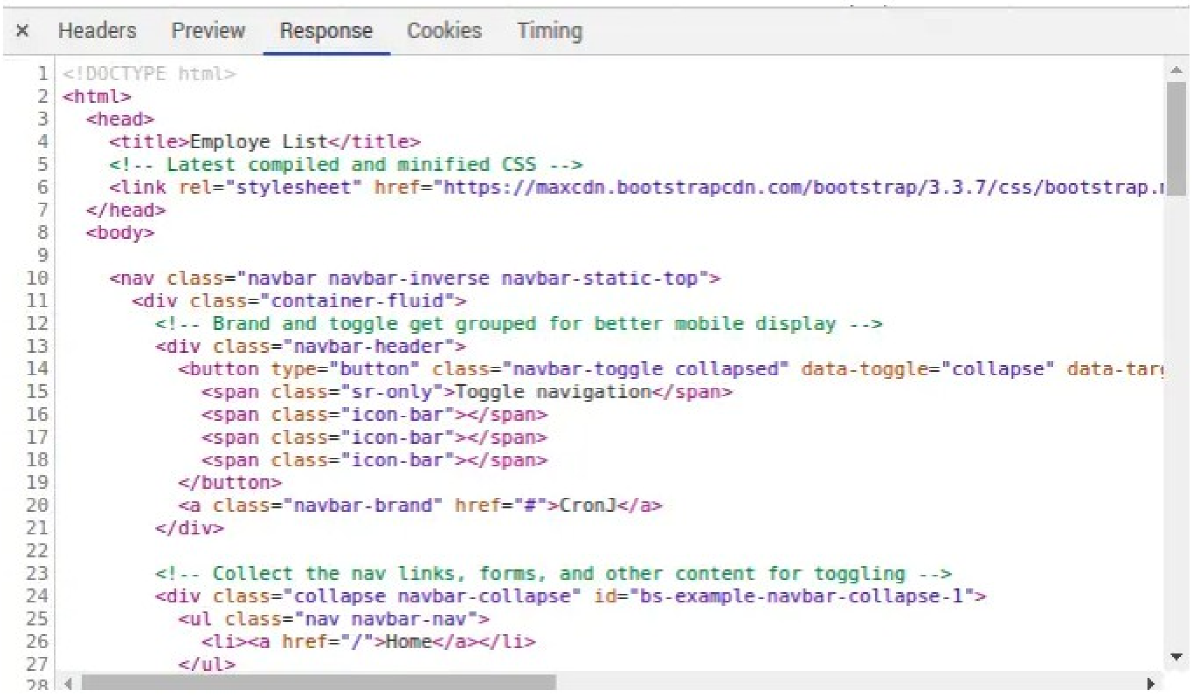
Status line:- It indicates the HTTP version, a status code, and a status message.

Response Header:- There can be one or more response header lines and they are used to pass additional information to the client from the server.



```
▸ General
▾ Response Headers     view source
    Connection: keep-alive
    Content-Length: 4189
    Content-Type: text/html; charset=utf-8
    Date: Tue, 09 Jul 2019 12:40:25 GMT
    ETag: W/"105d-1dHjy2Vd4mFw8MRKNZgzpPSpvbM"
    X-Powered-By: Express
▸ Request Headers (9)
```

Response Body:- The response body contains the actual content of the response (e.g., HTML, JSON, images). This contains the resource or data demanded by the client. If the request is unsuccessful then the response body contains the reason for the error, it may also contain the steps to be done by the client to complete the request successfully.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Employe List</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
  </head>
  <body>

    <nav class="navbar navbar-inverse navbar-static-top">
      <div class="container-fluid">
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-tar
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="#">CronJ</a>
        </div>

        <!-- Collect the nav links, forms, and other content for toggling -->
        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
          <ul class="nav navbar-nav">
            <li><a href="/">Home</a></li>
          </ul>
```

Creating Dynamic Contents

Dynamic content in web context refers to web page elements that change or are generated in real time based on various factors, such as user data inputs ,user interactions, or server-side changes. Unlike static content, which remains the same for every user and does not change unless manually updated, dynamic content is customizable and personalized.

Dynamic content is content that adapts based on input, time, or other variables, offering a more personalized and interactive web experience

Examples of dynamic content include:

1. **User-specific content**: Things like login information, personalized greetings, or customized product recommendations based on user behavior or preferences.
2. **Real-time data**: Stock prices, sports scores, or live chat updates that change frequently or continuously.
3. **Interactive elements**: Forms, filters, or search results that adjust and refresh based on user input without requiring the whole page to reload.
4. **Content from a database**: Articles, blog posts, or product listings that pull data from a backend server or database based on certain queries or parameters.

Technologies that make dynamic content possible often involve JavaScript (for front-end interactivity) and server-side programming languages like PHP, Python, Ruby, or Node.js (for backend data processing and generation).

Using control flow to control dynamic content generation

Dynamic content in the context of HTML and the World Wide Web refers to web content that constantly or regularly changes based on user interactions, timing and other parameters that determine what content is delivered to the user. This means that the content of the site may differ for every user because of different parameters. Facebook is an excellent example of a site that delivers dynamic content, as every user gets different content based on their friends and social interactions, although the layout stays generally the same.
This could be different text, video, advertisements or even an entirely different layout and color. Any element in a page which contains movement and can change over time may be considered as dynamic content.

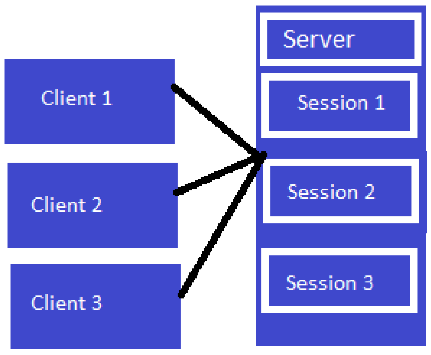
There are two ways to provide dynamic content:

1. **Using client-side scripting** and frameworks such as JavaScript, AJAX and Bootstrap to change the UI behavior within a specific Web page in response to specific user actions and timings. This gives dynamic behavior to the UI presentation. This is normally used in Web applications and interactive websites.
2. **Using server-side scripting** and processing to dynamically form both the layout and content to be delivered to the user based on parameters such as the user's location, time of day, browser being used or user preferences. Some good examples of this are social networking sites and content delivery sites. Social networking sites such as Facebook and Twitter provide entirely different content per user because of the difference of their connections and subscribed services, while sites like YouTube and Amazon provide dynamic content based on user-specific preferences based on past purchases or views; the server gives suggested items or content that the user may like based on historical data.

Sessions and State

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.
We all know that the web uses the HTTP protocol and the HTTP protocol is a stateless protocol; in other words, when a client sends a request to the server, an instance of the page is created and the page is converted to HTML format and then the server provides the response and then the instance of the page and the value of the control are destroyed. So if we have a requirement to store the values of the controls and pass them into another web form then a State Management Technique is used.
Session is a State Management Technique. A Session can store the value on the Server. It can support any type of object to be stored along with our own custom objects. A session is one of the best techniques for State Management because it stores the data as client-based, in other words the data is stored for every user separately and the data is secured also because it is on the server.



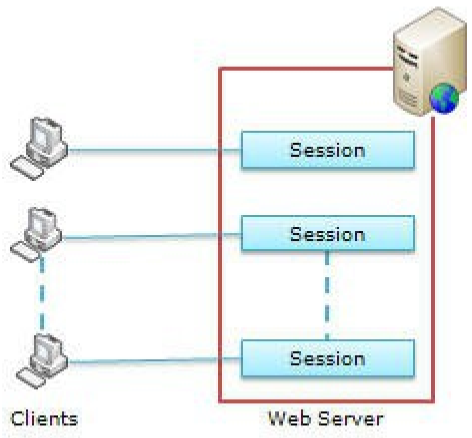


Fig: For every client, session data is stored separately

State management using session is one of the best ASP.NET features, because it is secure, transparent from users, and we can store any kind of object in it. Along with these advantages, sometimes session can cause performance issues in high traffic sites because it is stored in server memory and clients read data from the server. Now let's have a look at the advantages and disadvantages of using session in our web applications.

Advantages:

- It helps to maintain user state and data all over the application.
- It is easy to implement and we can store any kind of object eg. Database, dataset etc.
- Stores client data separately.
- Session is secure and transparent from the user.

Disadvantages:

- As session is stored on the server memory, it is not advisable to use session state when there is a large volume of data.
- With the use of session state, it will affect the performance of memory, because it is stored in server memory until you destroy the state
- The problem with sessions is that when you close your browser you also lose the session so, if you had a site requiring a login, this couldn't be saved as a session like it could as a cookie, and the user would be forced to re-login every time they visit.

Storing and retrieving values from Session
Starting a Session : `session_start();`
Storing a value: `$_SESSION['user'] = 'JohnDoe'` stores the value in the session.
Retrieving a value: `$_SESSION['user']` retrieves the stored value.
Removing a session value: `unset($_SESSION['user'])` removes a specific session variable.
Destroying the session: `session_destroy()` removes all session data.

```
<?php
      session_start();

      // Store a value in the session
      $_SESSION['user'] = 'JohnDoe';

      // Retrieve the value from the session
      if (isset($_SESSION['user'])) {
            echo "Hello, " . $_SESSION['user'];
      } else {
            echo "Hello, Guest!";
      }

      // To remove a session value
      unset($_SESSION['user']);

      // To destroy the session
      session_destroy();
?>
```

Session ID
A session ID is a unique string of characters (often generated by the server) that is sent between the client (usually a web browser) and the server to track the session.
When a client communicates with a server, only the session ID is transmitted between them. When the client requests for data, it looks for the session ID and retrieves the corresponding data. This is done in the following steps:

- Client hits the web site and information is stored in the session.
- Server creates a unique session ID for that client and stores it in the Session State Provider.
- The client requests for some information with the unique session ID from the server.
- Server looks in the Session Providers and retrieves the serialized data from the state server and type casts the object.

Take a look at the pictorial flow:

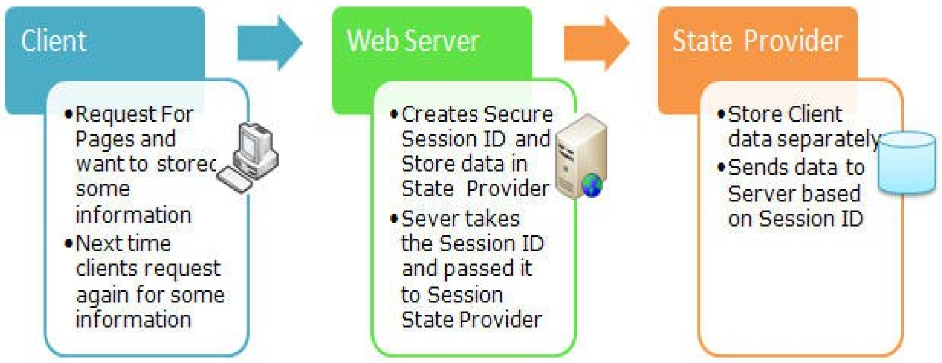

Client
•Request For Pages and want to storec some information
•Next time clients request again for some information
Web Server
•Creates Secure Session ID and Store data in State Provider
•Sever takes the Session ID and passed it to Session State Provider
State Provider
•Store Client data separately
•Sends data to Server based on Session ID

Fig: Communication of client, web server, and State Provider

Error Handling

Error handling in programming refers to the process of anticipating, detecting, and responding to runtime errors or unexpected conditions that may arise during the execution of a program. It's a critical aspect of writing reliable and robust software because it helps ensure that a program can gracefully handle issues like invalid user input, file not found, network failures, or mathematical errors like division by zero.

Types of Errors

1. **Syntax Errors:** These occur when the code violates the syntax rules of the programming language. These errors are usually caught by the compiler or interpreter before the program runs.
2. **Runtime Errors:** These happen during the execution of the program, like trying to divide by zero or accessing a variable that hasn't been initialized. These types of errors often result in program crashes or abnormal behavior.
3. **Logical Errors:** These are errors in the program's logic. The program may run without crashing but produce incorrect results.

Techniques for Error Handling

1. **Try-Catch Blocks:** A common way to handle errors in many programming languages is to use try and catch blocks (or except in Python). In this approach, we write the potentially error-prone code inside the try block and then catch and handle specific errors in the catch block.
   Example:
   ```
   try {
       $value = 10 / 0;  // Division by zero will throw an exception
   } catch (Exception $e) {
       echo 'Caught exception: ', $e->getMessage();  // Handle exception
   }
   ```

2. **Throwing/Catching Exceptions:** Instead of letting the program crash, exceptions allow the program to throw errors with specific messages or codes that can be caught and handled elsewhere in the program.
   We can manually throw exceptions using the throw keyword. This is useful for custom error conditions.
   Example:
   ```
   function divide($a, $b) {
         if ($b == 0) {
               throw new Exception('Division by zero');
         }
         return $a / $b;
   }

   try {
         echo divide(10, 0);
   } catch (Exception $e) {
         echo 'Error: ', $e->getMessage(); // Catch & display exception
   }
   ```

3. **Custom Error Handlers :** PHP also allows you to set custom error handlers using the set_error_handler() function. This allows you to define your own logic to handle errors in a more flexible way than the default PHP error handler.

```
// Custom error handler function
function customErrorHandler($errno, $errstr, $errfile, $errline) {
      echo "Error [$errno]: $errstr in $errfile on line $errline\n";
      // Log the error or perform other actions here
}

// Set the custom error handler
set_error_handler('customErrorHandler');

// Example of an error
echo $undefined_variable;  // This will trigger custom error handler
```

Architecting Web Application

What is App Application Architecture

Web application architecture is a blueprint of simultaneous interactions between components, database instances, middleware systems, user interfaces, and servers in an application. It can also be described as the layout that logically defines the connection between the server and client-side for a better web experience.

Understanding Web Application Architecture and Its Role in Performance

When a user enters a URL and presses "Go," the browser locates the server hosting the website and sends a request for the specific page. The server responds by sending the necessary files back to the browser. The browser then processes these files and displays the requested page to the user. This process happens almost instantly, allowing users to interact with the website without noticeable delay. If websites were slow or unresponsive, users would quickly abandon them.

At the core of this process is the code that the browser parses and executes. This code may contain instructions that tell the browser how to respond to different user inputs. Web application architecture refers to the structure and design of the entire web-based software application, including its subcomponents and interactions with external services.

Efficient web application architecture is crucial for ensuring performance, scalability, security, and robustness. Given that most global network traffic relies on web-based communication, it is vital that web applications function smoothly to meet the needs of users across a wide range of devices and platforms.

Best Practices for Good Web Application Architecture

You may have a working app, but it also needs to have good web architecture. Here are several attributes necessary for good web application architecture:

1. **Modular Design**: Split the application into distinct, independent modules for easier management and scalability.
2. **Scalability**: Design the system to handle increasing user traffic and data growth without degrading performance.
3. **API-First Approach**: Use RESTful APIs or GraphQL for flexible communication between the frontend and backend.
4. **Responsive Design**: Ensure the application works well on various devices, adapting to different screen sizes.
5. **Performance Optimization**: Improve load times by minimizing resources, lazy loading, and using caching techniques.
6. **Security Measures**: Implement strong authentication, encryption, input validation, and protection against common vulnerabilities like XSS and SQL injection.
7. **Database Management**: Use appropriate database designs and indexing to ensure data integrity and fast queries.
8. **Microservices Architecture**: Split functionality into smaller, independent services to allow easier scaling and fault isolation.
9. **Maintainable Code**: Write clean, reusable, and well-documented code, following best practices like DRY and KISS.
10. **CI/CD Pipelines**: Automate testing, building, and deployment processes to ensure quick and reliable releases.
11. **Monitoring & Logging**: Track application performance and errors in real-time with

monitoring tools and centralized logging.

12. **Cross-Browser Compatibility**: Ensure the application performs consistently across different browsers and platforms.
13. **Content Delivery Networks (CDNs)**: Use CDNs to reduce server load and speed up content delivery to users globally.
14. **Error Handling**: Gracefully manage and display errors, ensuring the system remains stable and informative during issues.
15. **User-Centric Design**: Prioritize user experience with clear navigation and responsive interfaces to meet user needs effectively.

Types of Web Application Architecture

1. Single Page Application Architecture
2. Microservice Architecture
3. Serverless Architecture
4. Progressive Web Applications

Using Tag Libraries

Tag libraries are sets of custom tags that can be used in JavaServer Pages (JSP) to enhance the functionality and structure of a web application. They allow developers to encapsulate complex logic into reusable components, which can be embedded within JSPs. This promotes code reuse, clean separation of concerns, and easy maintenance.

What are Tag Libraries?

A tag library is a collection of custom JSP tags, each corresponding to some Java code (a handler class) that executes certain behavior. These custom tags can perform tasks like manipulating data, rendering content, and interacting with back-end services, while abstracting away the complexity.

The most common tag library used in JSP is JSTL (JavaServer Pages Standard Tag Library), which provides a set of standard tags to perform common tasks like iteration, conditionals, and database interactions.

Key Benefits:

1. **Code Reusability**: Custom tags can be written once and reused across multiple JSP pages.
2. **Separation of Concerns**: Tag libraries promote separating business logic from presentation logic.
3. **Simplified Syntax**: Custom tags can reduce the amount of Java code written in JSP files.

References
- https://litslink.com/blog/web-application-architecture
- https://www.clickittech.com/software-development/web-application-architecture/
- https://medium.com/@gwenilorac/layout-of-web-applications-795b3e8e4c1b
- https://www.simform.com/blog/web-application-architecture/
- https://www.techtarget.com/whatis/definition/server
- https://www.paessler.com/it-explained/server
-
-