**What is Server:-**

* A server is a system that provides services or resources to other computer or devices over a network.
* Server manages and process request from clients and respond to them, usually through a network protocol as HTTP,FTP,SMTP.

**What is server side Programming:-**

* It refers to code that runs a server rather than client machine.
* It involves creating programs that handle requests from clients, process it and then generate a response that is sent back to content.

e.g. PHP, Java (Servlets/JSP),Node.js, Python (Django, Flask)

**What is Servlet?**

🡪A servlet is a java program that runs on a web server or application servers, handling requests from web clients and responding back to them.

🡪It acts as a middle layer between client request and server response.

🡪It is used to develop dynamic web applications. They can:-

* Receive data from a client
* Process the data
* Generate a response to send back to the client

**How Servlet Works :-**

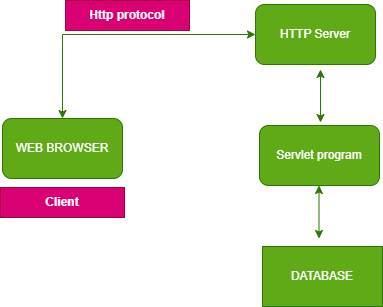
1>Client Request: A user enters a YUL or submit a form.

2>Request Handling: The Servlet container (Apache Tomcat) forwards the request to the corresponding servlet.

3>Processing: The Servlet reads the request, processing the data and generate a response.

4>Response: The servlet sends back the response (typically HTML) to the client.

**Simple Architecture of Servlet:-**



CODE EXAMPLE (First\_Program.java) Folder:-(Servlet\_ajava)

**serialVersionUID declaration in code:-**

* It’s a unique identifier that is used by Serialization mechanism.
* Serialization is the process of object to convert byte stream.
* Deserialization is the process of byte stream convert to object.
* It’s basically used to control the InvalidClassException if we made any change to class.

**What is CGI (Comman Gateway Interface):-**

🡪It’s a standard protocol that allows web servers to interact with external programs or scripts to dynamically generate content.

**Disadvantage of CGI :-**

1>**Performance:-** Since CGI spawn new process for every request, if traffic increases then it become slow.

2>**Scalability:-** For each request ir=t start new process and web server and there limited number to start new process.

3>It uses platform dependent language e.g. python,perl

**Advantage of Servlet over CGI :-**

1>**Efficent:-** Servlets handles requests using thread instead of creating a new process for every request. This Provide more performance.

2>**Cross-Platform:-** For Written in java that’s why servlets are portable across platforms.

3>**Secure:-** It’s more secure than CGI.

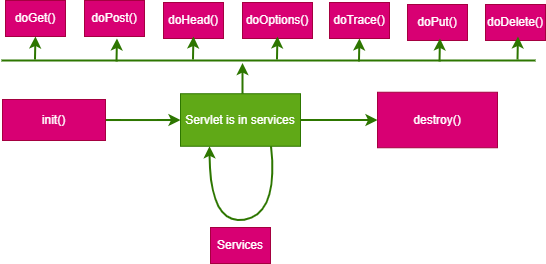
4>**Robust-API:-** Servlets have build-in support for server tracking, cookies and authentication.

**WEB Terminology :-**

🡪It covers the essential concepts involved in web communication and website development.

|  |  |
| --- | --- |
| Static Website | Dynamic Website |
| 1>A static website consists of webpages with fixed content. | 1>In dynamic website content can change based on user interaction or other factor. |
| 2>No-server side scripting is used. | 2>Server side Scripting may be used. |
| 3>Ideal for small website where content doesn’t changed frequently. | 3>Ideal for big scale project where content changed frequently. |
| 4>Faster to load because the server doesn’t need to process data. | 4>Requires a server to process user inputs, database queries then generate the output which makes it little bit slow compare static website to load. |
| 5>Pages are created using basic HTML and CSS | 5>Pages are often generated by server-side technologies like JAVA,PHP, Python, etc. |

**Life of Cycle of Servlet:-**

****

1>init():- It is called when servlet is created.It called once for each servlet

Code syntax:- public void init() throws ServletException{

//intialization code

}

2>Service():- It is main method that performs actual task.

It checks the HTTP request type(Get,Post,etc) and in response calls the doGet,doPost,etc.

🡪suppose, client(request is sending)-->Servlet container(It sends the request to Service method)-->Service method(Handles the request)

Code syntax:- public void Service(ServletRequest req,ServletResponse res)throws ServletException,IOException{

//code

}

public void doGet(HTTPServletRequest request,HTTPServletResponse response)throws ServletException,IOException{

//code }

public void doPost(HTTPServletRequest request,HTTPServletResponse response)throws ServletException,IOException{

//code

}

3>destroy():- Only called once at the end of servlet life cycle.Basically its used to close the connection to database or close cookies.Basically it contains clean up code

Code syntax:- public void destroy(){

//clean up code

}

CODE EXAMPLE (Servlet\_LC.java) Folder:-(Servlet\_ajava)

**NOTE:-(TO learn to know about it in details go to the topic Servlet lifecycle in SC)**

**HTTP :-**

🡪It stands for Hyper text transfer Protocol

🡪It’s foundation of data communication on the web.

🡪It’s an application layer protocol that enables the transfer of data between a web browser and web server.

**HTTP/1.1:**Most widely used, persistent for connection.

**HTTP/2**:Faster,supports multiplexing

**HTTP/3**:Introduce even more speed and security requirements.

**HTTP Requests:-**

🡪An Http request is a message sent by a client a server to request some resources.

🡪The request has a specific structure.

1. Request Line:-

(i**)Method**:-Defines the what client wants to do (Get, Post, etc)

(ii)[**URL:-** Specifies](URL:-%20Specifies) the resources being requested

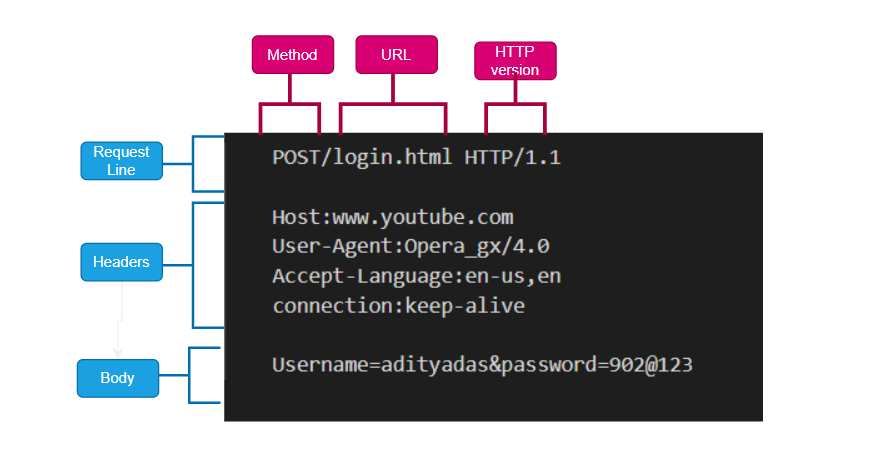
(iii)**HTTP version**:- Defines the version of http being used

1. Headers: Provides the metadata about the request
2. Body: Contains the actual data being send

* **Common Http Request Method:-**

1. **Get:-** Used to request data from the server. Typically, it’s doesn’t modify the state on the server. Data is send in the URL.
2. **Post:-** Used to send data to the server. It’s usually used when submitting the data and the data is included in the body of the request.
3. **Put:-** Replace an existing resource or creates a new one.
4. **Delete:-** Deletesthe specified resources.
5. **Options:-** Describes the communication options for the target resource.

* **Example of Http Requests:-**



**GET vs POST:-**

🡪 Both GET and POST methods are used to request data to a server, but they are used differently.

* GET Method:-
  + Used to retrieve data
  + Data is passed in the URL as query parameter
  + Not secure for sensitive data, since data is visible in URL
  + Limited by URL length

Code EXAMPLE:- (GET.java) Folder:-(Servlet\_ajava)

* Post Method:-
  + Used to submit data (such as form data)
  + Data is sent in the body of request, not in the URL
  + More secure, specially for retrieve data
  + Not have limit on amount of data send

Code EXAMPLE:- (POST.java,form.html) Folder:-(Servlet\_ajava)

|  |  |  |
| --- | --- | --- |
|  | GET Method | Post Method |
| Purpose | Retrieve data | Submit data |
| Data in URL | Yes | No |
| Security | Less Secure | More Secure |
| Data Limited to transfer | Limited in Length | No data limited |
| Idempotency | Safe to repeat without any changes | Can change the server state |

**Servlet Container(SC):-**

🡪A servlet container is a part of a web server that provides the environment for running JAVA servlets and manage their lifecycle.

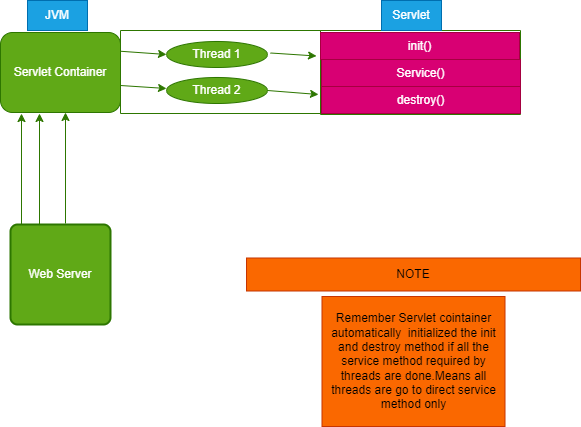
🡪It acts as an intermediary between client and server side component.

* **Responsibility of SC:-**
  + **Lifecycle management:-**The servlet contains manages the lifecycle of servlet.
  + **Request Handling:-**The servlet contains manages the lifecycle of. It receives HTTP request from clients, process them and forward them to the appropriate servlet for handling.
  + **Response Generation:-** After the servlet process the request, the container handles the response generation and back to the client.
  + **Concurrency Handling:-** The servlet container allows multiple clients to interact with servlets simultaneously by creating separate threads for each request.
  + **Session management:-** It manages user session, allowing servlets to maintains state across multiple request from the same client.

* **Common Servlets Container:-**
  + **Apache Tomcat:-**The most Widely used Servlet Container which implements the java servlet.
  + **Jetty:-** A lightweight servlet container that can be embedded in application.
  + **GlassFish:-** An application server that supports JAVA EE specification, including servlets.
* **Servlet life cycle in Container:-**

🡪The Servlet lifecycle managed by the servlet container consists of the following phases:-

1. **Loading and Instantiation:-** The servlet container loads the servlet class and creates an instance of it.
2. **Initialization:**- The container calls the init() method of the servlet to initialize it. This method is called once during Servlets’ s lifecycle.
3. **Request Handling:**- The Container invokes the service() method for each request to the servlet. The Servlet process the request and generate a response.
4. **Destruction:**- When the servlet is no longer needed, the container calls the destroy() method to release resource and performs cleanup.



CODE EXAMPLE (Servlet\_LC.java) Folder:-(Servlet\_ajava)

**Servlet API:-**

🡪The **Servlet API** is a collection of interfaces and classes that define the rules and mechanisms for developing Java-based web applications.

🡪It allows developers to create servlets, which are Java programs that run on a server and handle client requests (usually over HTTP), process them, and generate responses.

**1. Servlet Interface**

At the core of the Servlet API is the javax.servlet.Servlet interface. Every servlet class you write must implement this interface, which defines methods that manage the lifecycle of a servlet.

**2. HttpServlet Class**

The HttpServlet class is a subclass of GenericServlet, and it provides HTTP-specific methods. In most web applications, servlets extend HttpServlet, as they often deal with HTTP requests (GET, POST, etc.).

**3. ServletRequest Interface**

The ServletRequest interface provides data about the client's request, like parameter values, request headers, and more.

**4. HttpServletRequest Interface**

This is a specialized version of ServletRequest, designed specifically for HTTP requests. It includes methods that deal with HTTP-specific data.

**5. ServletResponse Interface**

This interface provides methods to formulate the response that will be sent back to the client.

**6. HttpServletResponse Interface**

This interface extends ServletResponse and is used to work with HTTP-specific functionality.

**7. ServletConfig Interface**

The ServletConfig interface provides configuration information for the servlet. It allows you to retrieve initialization parameters and other configuration data.

**8. ServletContext Interface**

The ServletContext interface represents the entire web application. It provides methods for logging, sharing data between servlets, and accessing web application resources.

**9. Life Cycle of a Servlet**

**Servlet Interface:-**

🡪The Servlet interface is the core interface for all servlets in Java. Every servlet, either directly or indirectly, implements this interface.

🡪It defines the lifecycle methods of a servlet and provides a way for a servlet to handle requests and responses.

* **METHODS OF SERVLETS INTERFACE**

🡪 The Servlet interface defines **five key methods** that every servlet must implement. These methods manage the servlet's lifecycle, from its initialization to its destruction, and handle client requests during its active state.

1. **init(ServletConfig config)** – Initialization of the servlet.

The init() method is called when the servlet is first initialized. This happens once during the servlet's lifecycle, and it's used to perform any necessary setup before handling requests, such as initializing resources (e.g., database connections, loading configuration files).

1. **service(ServletRequest request, ServletResponse response)** – Handling requests.

This method is the heart of the servlet and is called by the servlet container to handle every incoming request. The servlet processes the request and then generates a response.

* **ServletRequest**: Represents the client’s request and allows the servlet to retrieve data sent from the client, such as form parameters or headers.
* **ServletResponse**: Represents the response sent back to the client, such as HTML content or binary data.

1. **destroy() –** Destruction and cleanup of resources.

The destroy() method is called by the servlet container when the servlet is about to be unloaded, typically when the server is shutting down or when the servlet is being replaced. This method allows the servlet to release any resources (like closing database connections or clearing memory) before the servlet instance is destroyed.

1. **getServletConfig() –** Retrieving servlet configuration.

The getServletConfig() method returns the ServletConfig object associated with the servlet. This object holds the initialization parameters and provides access to the servlet’s configuration.

1. **getServletInfo()** – Providing metadata about the servlet.

The getServletInfo() method returns information about the servlet, such as its version, author, or other descriptive metadata. It’s not commonly used, but it can be helpful for documentation or debugging purposes.

CODE EXAMPLE (Servlet\_Interface.java) Folder:-(Servlet\_ajava)

**Servlet Generating HTML Output:-**

Java Servlets are server-side components that handle requests and responses in a web application.

One of their key capabilities is generating dynamic HTML output based on client requests.

CODE EXAMPLE (Time\_servlet.java) Folder:-(Servlet\_ajava)

**Types of Servlets: GenericServlet and HttpServlet:-**

Java servlets are powerful components used for building web applications that can handle dynamic requests and responses. The **Servlet API** provides two core classes for building servlets:

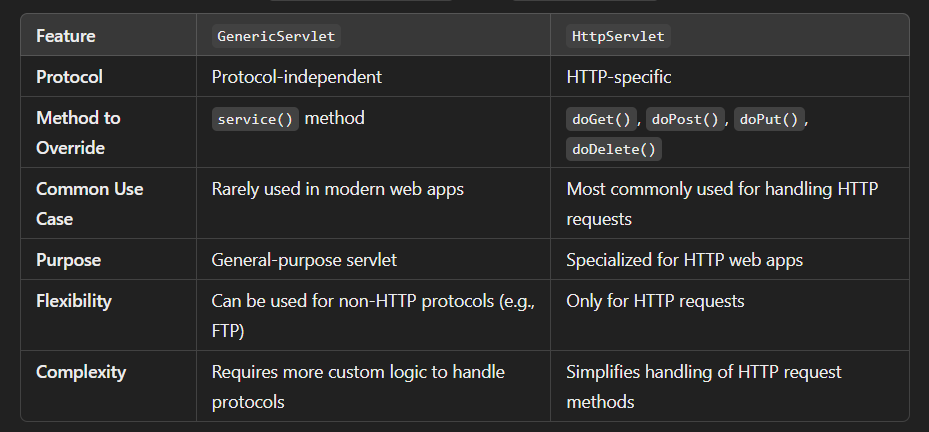
1. **GenericServlet**
2. **HttpServlet**

1.The **GenericServlet** class is an abstract class that implements the **Servlet** interface, providing a generic and protocol-independent implementation of a servlet. It acts as a base class for servlets that don't depend on the HTTP protocol specifically. While rarely used for actual HTTP-based web applications, it can theoretically handle any protocol, such as FTP, SMTP, etc.

CODE\_EXAMPLE (MyGeneric\_servlet.java) Folder(Servlet\_ajava)

2. The **HttpServlet** class extends **GenericServlet** and is specifically designed for handling HTTP requests and responses. Since the majority of web applications use HTTP as their communication protocol, **HttpServlet** is the most commonly used servlet class.

CODE\_EXAMPLE (Http\_servlet.java,) Folder(Servlet\_ajava)



**Server Collabrations:-**

**1>RequestDispatcher:-**

The RequestDispatcher interface provides a way to forward a request from one servlet to another servlet or resource (like JSP or HTML) on the server-side. It allows servlets to collaborate by passing control from one to another without the client’s knowledge.

**Methods:**

* **forward()**: Forwards the request from one servlet to another resource (another servlet, JSP, or HTML).
* **include()**: Includes content from another resource in the response of the current servlet.

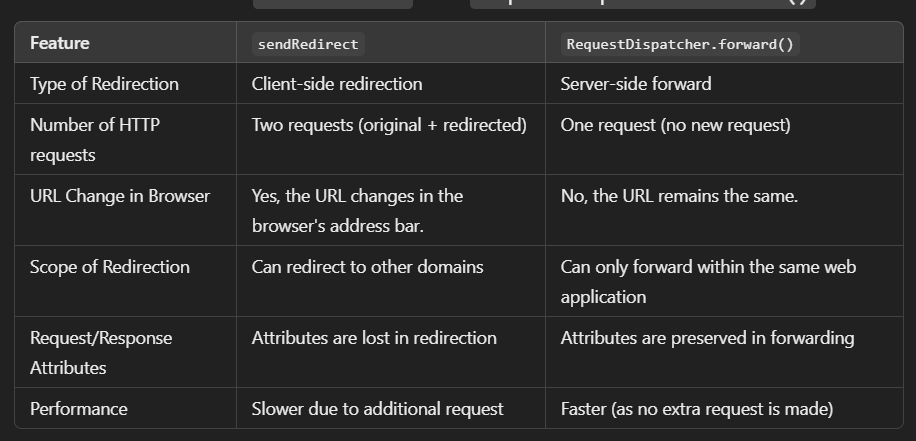
CODE\_EXAMPLE (Servlet1\_s.java, Servlet2\_s.java,RD1.html) Folder(Servlet\_ajava2)

**2>sendRedirect:-**

sendRedirect is a method in the HttpServletResponse interface that is used to redirect the client's browser to another URL.

This method is a part of the **HTTP** protocol, and it essentially tells the client to make a new request to a different URL, allowing the server to handle different requests by redirecting them as needed.

CODE\_EXAMPLE (Redirect\_1.java,sample1.html,Sample2.html,Login.html Folder(Servlet\_ajava2)

\

**3>ServletConfig:-**

It is an interface provided by the Java Servlet API to pass configuration information to a servlet at the time of initialization.

* **Key features of ServletConfig:**

1. **Initialization Parameters**: You can use ServletConfig to pass configuration parameters (like database connection settings, file paths, etc.) to a servlet during initialization.
2. **Servlet Context Access**: Through ServletConfig, you can get access to the ServletContext object, which provides information about the servlet environment.
3. **Limited to One Servlet**: The configuration parameters provided through ServletConfig are specific to the servlet that it’s associated with and cannot be shared among servlets.

* **ServletConfig Methods**
* **getInitParameter(String name)**: Returns a string containing the value of the named initialization parameter, or null if the parameter does not exist.
* **getInitParameterNames()**: Returns an enumeration of the names of the servlet's initialization parameters.
* **getServletContext()**: Returns the ServletContext object, which gives information about the environment in which the servlet is running.
* **getServletName()**: Returns the name of the servlet instance.

CODE\_EXAMPLE(ConfigExampleServlet.java, web.xml) Folder(Servlet\_ajava3)

**ServletContext:-**

It is a shared object provided by the servlet container (e.g., Apache Tomcat) that gives servlets access to configuration details, context-level attributes, and resources that are global to the web application.

There is only one ServletContext per web application, and it is used to:

1. **Share Information**: It allows servlets to share information (attributes) across the entire web application.
2. **Access Web Application Resources**: It provides access to configuration parameters, files, resources, and metadata of the application.
3. **Inter-Servlet Communication**: Servlets can use it to share data or call other servlets in the same application.

* **Methods in ServletConfig:-**

1. **getInitParameter(String name)**: Returns the value of the named context-wide initialization parameter from web.xml, or null if the parameter does not exist.

2. **getInitParameterNames()**: Returns an Enumeration of the names of all the context-wide initialization parameters.

3. **getServletContextName()**: Returns the name of the web application as defined in web.xml.

4.**setAttribute(String name, Object object)**: Allows servlets to store an attribute in the context, which can be accessed by any servlet in the application.

5. **getAttribute(String name)**: Retrieves an attribute from the context, which was previously set using setAttribute().

6. **removeAttribute(String name)**: Removes the attribute stored under the given name in the context.

7. **getContext(String uripath)**: Returns a ServletContext object that corresponds to another web application on the server, allowing cross-context communication .

CODE\_EXAMPLE(ServletContextExample.java) Folder(Servlet\_ajava4)

**Session tracking:-**

It is a crucial aspect of web development, enabling servers to maintain the state of user interactions across multiple requests. In Java Servlets, several methods are available for session tracking, including Cookies, Hidden Form Fields, URL Rewriting, and HttpSession.

**1>Cookies:-**

🡪Cookies are a fundamental component of web technology that allow servers to store small amounts of data on a user's computer, enabling stateful sessions in stateless HTTP.

🡪They are primarily used for tracking user sessions, storing user preferences, and maintaining user authentication.

* **Types of Cookies**

1. **Session Cookies**:
   * **Lifetime**: Temporary cookies that are deleted once the browser is closed.
   * **Use Case**: Used to maintain the session of a user while navigating through the website.
2. **Persistent Cookies**:
   * **Lifetime**: Stored on the user's device for a set period or until manually deleted by the user.
   * **Use Case**: Used to remember user preferences or authentication for future visits, such as remembering a user's language choice or keeping them logged in.
3. **Secure Cookies**:
   * **Security**: Can only be transmitted over HTTPS, enhancing security by preventing interception.
   * **Use Case**: Used for sensitive data such as session tokens.
4. **HttpOnly Cookies**:
   * **Security**: Not accessible via JavaScript, which helps mitigate the risk of cross-site scripting (XSS) attacks.
   * **Use Case**: Used to store session identifiers securely.

CODE\_EXAMPLE(Read\_cookie.java,Create\_cookie.java) Folder(Servlet\_ajava4)

**2>Hidden form Field:-**

🡪Hidden form fields are a useful feature in web forms, allowing developers to include additional information that the user cannot see or modify.

🡪They are represented in HTML using the <input type="hidden"> element.

* **Characteristics of Hidden Form Fields**

1. **Visibility**:
   * Hidden fields do not display any data to the user. They are entirely hidden from the user interface, which makes them ideal for transmitting data that should not be manipulated by the user.
2. **Data Transmission**:
   * When the form containing hidden fields is submitted, the data from these fields is sent to the server as part of the HTTP request, just like visible fields.
3. **Data Type**:
   * The value of hidden fields is typically a string.

* **Use Cases for Hidden Form Fields**

1. **User Session Management**:
   * Hidden fields can store session IDs or tokens that help identify user sessions without requiring the user to input this information.
2. **Form State Management**:
   * They can keep track of the user's progress through multi-step forms, storing values that may not need to be shown on every step.
3. **Security Measures**:
   * Hidden fields can carry important identifiers (like user IDs) that ensure the server can correctly associate data with the right user, though these should be carefully validated to prevent tampering.
4. **Pass Data Between Pages**:
   * They are useful for passing data between different forms or pages without displaying the information to the user, such as keeping track of previous selections.

CODE\_EXAMPLE (HiddenForm.java,HTML\_Hidden\_form.html) Folder(Servlet\_ajava4)

**3>URL Rewriting:-**

URL Rewriting is a technique used in web applications to change the URL structure of a webpage.

The goal is to make URLs more readable for users and to pass additional data (like user information) to the server without using cookies.

Using URL Rewriting

1. **User-Friendly URLs**: Make URLs easier to read and remember.
   * Example: Instead of http://example.com/product?id=123, a more friendly URL could be http://example.com/product/123.
2. **Session Tracking**: Keep track of user sessions without using cookies.
3. **Hiding Sensitive Information**: Avoid showing sensitive data directly in the URL.

CODE\_EXAMPLE (URL\_Rewrite.java, url\_rewrite.html) Folder(Servlet\_ajava4)

**4>HttpSession:-**

It a part of the Java Servlet API that provides a mechanism for maintaining state across multiple requests from the same client.

Since HTTP is a stateless protocol, HttpSession allows developers to store and retrieve data for a particular user session, enabling features like user authentication, shopping carts, and personalized user experiences.

* **Features of HttpSession**

1. **Session Creation**: A new session is created when a user first accesses a web application. Each session is uniquely identified by a session ID.
2. **Session Persistence**: Data stored in the session remains available across multiple requests until the session expires or is invalidated.
3. **Server-Side Storage**: Session data is stored on the server side, which helps protect sensitive information.
4. **Time-Based Expiration**: Sessions can be configured to expire after a certain period of inactivity.

CODE\_EXAMPLE (Login\_servlet.java,Logout.java,Welcome.java,login.html,welcome.jsp) Folder(Servlet\_ajava5)

**Crud operations in Servlets:-**

🡪Doing crud operation using jdbc and html forms using servlet technology.

CODE\_EXAMPLE(AddUserServlet.java, DeleteUser.java, SelectUser.java, UpdateUser.java) Folder(S\_JDBC)