Experiment Title: Write a program to manage session in ASP.Net

Objective:

To learn and implement management of sessions in ASP.Net

Theory:

ASP.NET State Management

A new instance of the Web page class is created each time the page is posted to the server. In traditional Web programming, this would typically mean that all information associated with the page and the controls on the page would be lost with each round trip. For example, if a user enters information into a text box, that information would be lost in the round trip from the browser or client device to the server.

To overcome this inherent limitation of traditional Web programming, ASP.NET includes several options that help you preserve data on both a per-page basis and an application-wide basis. These features are as follows:

- View state
- Control state
- Hidden fields
- Cookies
- Query strings
- Application state
- Session state
- Profile Properties

View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways. However, application state, session state, and profile properties all store data in memory on the server. Each option has distinct advantages and disadvantages, depending on the scenario.

Client-Based State Management Options

The following sections describe options for state management that involve storing information either in the page or on the client computer. For these options, no information is maintained on the server between round trips.

(a) View State

The ViewState property provides a dictionary object for retaining values between multiple requests for the same page. This is the default method that the page uses to preserve page and control property values between round trips.

When the page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field, or multiple hidden fields if the amount of data stored in the ViewState property exceeds the specified value in the MaxPageStateFieldLength property. When the page is posted back to the server, the page parses the view-state string at page initialization and restores property information in the page.

(b) Control State

Sometimes you need to store control-state data in order for a control to work properly. For example, if you have written a custom control that has different tabs that show different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips. The ViewState property can be used for this purpose, but view state can be turned off at a page level by developers, effectively breaking your control. To solve this, the ASP.NET page framework exposes a feature in ASP.NET called control state.

The ControlState property allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property.

(c) Hidden Fields

ASP.NET allows you to store information in a HiddenField control, which renders as a standard HTML hidden field. A hidden field does not render visibly in the browser, but you can set its properties just as you can with a standard control. When a page is submitted to the server, the content of a hidden field is sent in the HTTP form collection along with the values of other controls. A hidden field acts as a repository for any page-specific information that you want to store directly in the page.

Security Note

It is easy for a malicious user to see and modify the contents of a hidden field. Do not store any information in a hidden field that is sensitive or that your application relies on to work properly.

A HiddenField control stores a single variable in its Value property and must be explicitly added to the page. In order for hidden-field values to be available during page processing, you must submit the page using an HTTP POST command. If you use hidden fields and a page is processed in response to a link or an HTTP GET command, the hidden fields will not be available.

(d) Cookies

A cookie is a small amount of data that is stored either in a text file on the client file system or in-memory in the client browser session. It contains site-specific information that the server sends to the client along with page output. Cookies can be temporary (with specific expiration times and dates) or persistent.
You can use cookies to store information about a particular client, session, or application. The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information. The server can read the cookie and extract its value. A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in your application.

Security Note

The browser can only send the data back to the server that originally created the cookie. However, malicious users have ways to access cookies and read their contents. It is recommended that you do not store sensitive information, such as a user name or password, in a cookie. Instead, store a token in the cookie that identifies the user, and then use the token to look up the sensitive information on the server.

(e) Query Strings

A query string is information that is appended to the end of a page URL. A typical query string might look like the following example:

```
http://www.contoso.com/listwidgets.aspx?category=basic&price=100
```

In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "category" and the other called "price."
Query strings provide a simple but limited way to maintain state information. For example, they are an easy way to pass information from one page to another, such as passing a product number from one page to another page where it will be processed. However, some browsers and client devices impose a 2083-character limit on the length of the URL.

Security Note

Information that is passed in a query string can be tampered with by a malicious user. Do not rely on query strings to convey important or sensitive data. Additionally, a user can bookmark the URL or send the URL to other users, thereby passing that information along with it.

In order for query string values to be available during page processing, you must submit the page using an HTTP GET command. That is, you cannot take advantage of a query string if a page is processed in response to an HTTP POST command.

Server-Based State Management Options

ASP.NET offers you a variety of ways to maintain state information on the server, rather than persisting information on the client. With server-based state management, you can decrease the amount of information sent to the client in order to preserve state, however it can use costly resources on the server. The following sections describe three server-based state management features: application state, session state, and profile properties.

(f) Application State

ASP.NET allows you to save values using application state — which is an instance of the HttpApplicationState class — for each active Web application. Application state is a global storage mechanism that is accessible from all pages in the Web application. Thus, application state is useful for storing information that needs to be maintained between server round trips and between requests for pages.

Application state is stored in a key/value dictionary that is created during each request to a specific URL. You can add your application-specific information to this structure to store it between page requests.

Once you add your application-specific information to application state, the server manages it.

(g) Session State

ASP.NET allows you to save values by using session state — which is an instance of the HttpSessionState class — for each active Web-application session.

Session state is similar to application state, except that it is scoped to the current browser session. If different users are using your application, each user session will have a different session state. In addition, if a user leaves your application and then returns later, the second user session will have a different session state from the first.

Session state is structured as a key/value dictionary for storing session-specific information that needs to be maintained between server round trips and between requests for pages

You can use session state to accomplish the following tasks:

- Uniquely identify browser or client-device requests and map them to an individual session instance on the server.
- Store session-specific data on the server for use across multiple browser or client-device requests within the same session.
- Raise appropriate session management events. In addition, you can write application code leveraging these events.

Once you add your application-specific information to session state, the server manages this object. Depending on which options you specify, session information can be stored in cookies, on an out-of-process server, or on a computer running Microsoft SQL Server.

(h) Profile Properties

ASP.NET provides a feature called profile properties, which allows you to store user-specific data. This feature is similar to session state, except that the profile data is not lost when a user's session expires. The profile-properties feature uses an ASP.NET profile, which is stored in a persistent format and associated with an individual user. The ASP.NET profile allows you to easily manage user information without requiring you to create and maintain your own database. In addition, the profile makes the user information available using a strongly typed API that you can access from anywhere in your application.

You can store objects of any type in the profile. The ASP.NET profile feature provides a generic storage system that allows you to define and maintain almost any kind of data while still making the data available in a type-safe manner.

To use profile properties, you must configure a profile provider. ASP.NET includes a SqlProfileProvider class that allows you to store profile data in a SQL database, but you can also create your own profile provider class that stores profile data in a custom format and to a custom storage mechanism such as an XML file, or even to a web service.

Because data that is placed in profile properties is not stored in application memory, it is preserved through Internet Information Services (IIS) restarts and worker-process restarts without losing data. Additionally, profile properties can be persisted across multiple processes such as in a Web farm or a Web garden.

ASP.NET Session State

Use ASP.NET session state to store and retrieve values for a user.

Background

ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application. HTTP is a stateless protocol. This means that a Web server treats each HTTP request for a page as an independent request. The server retains no knowledge of variable values that were used during previous requests. ASP.NET session state identifies requests from the same browser during a limited time window as a session, and provides a way to persist variable values for the duration of that session. By default, ASP.NET session state is enabled for all ASP.NET applications.

Alternatives to session state include the following:

- Application state, which stores variables that can be accessed by all users of an ASP.NET application.
- Profile properties, which persists user values in a data store without expiring them.
- ASP.NET caching, which stores values in memory that is available to all ASP.NET applications.
- View state, which persists values in a page.
- Cookies.
- The query string and fields on an HTML form that are available from an HTTP request.

(i) Session Variables

Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property. In an ASP.NET page, the current session variables are exposed through the **Session** property of the **Page** object.

The collection of session variables is indexed by the name of the variable or by an integer index. Session variables are created by referring to the session variable by name. You do not have to declare a session variable or explicitly add it to the collection. The following example shows how to create session variables in an ASP.NET page for the first and last name of a user, and set them to values retrieved from TextBox controls.

C#

```
Session["FirstName"] = FirstNameTextBox.Text;
Session["LastName"] = LastNameTextBox.Text;
```

Session variables can be any valid .NET Framework type. The following example stores an ArrayList object in a session variable named `StockPicks`. The value returned by the `StockPicks` session variable must be cast to the appropriate type when you retrieve it from the SessionStateItemCollection.

C#

```
// When retrieving an object from session state, cast it to
// the appropriate type.
ArrayList stockPicks = (ArrayList)Session["StockPicks"];

// Write the modified stock picks list back to session state.
Session["StockPicks"] = stockPicks;
```

Note

When you use a session-state mode other than InProc, the session-variable type must be either a primitive .NET type or serializable. This is because the session-variable value is stored in an external data store.

(j) Session Identifiers

Sessions are identified by a unique identifier that can be read by using the SessionID property. When session state is enabled for an ASP.NET application, each request for a page in the application is examined for a SessionID value sent from the browser. If no SessionID value is supplied, ASP.NET starts a new session and the SessionID value for that session is sent to the browser with the response.

By default, SessionID values are stored in a cookie. However, you can also configure the application to store SessionID values in the URL for a "cookieless" session.

A session is considered active as long as requests continue to be made with the same SessionID value. If the time between requests for a particular session exceeds the specified time-out value in minutes, the session is considered expired. Requests made with an expired SessionID value result in a new session.

Security Note

SessionID values are sent in clear text, whether as a cookie or as part of the URL. A malicious user could get access to the session of another user by obtaining the SessionID value and including it in requests to the server. If you are storing sensitive information in session state, it is recommended that you use SSL to encrypt any communication between the browser and server that includes the SessionID value.

(i) Cookieless SessionIDs

By default, the SessionID value is stored in a non-expiring session cookie in the browser. However, you can specify that session identifiers should not be stored in a cookie by setting the **cookieless** attribute to **true** in the sessionState section of the Web.config file.

The following example shows a Web.config file that configures an ASP.NET application to use cookieless session identifiers.

```
<configuration>
  <system.web>
    <sessionState cookieless="true"
      regenerateExpiredSessionId="true" />
```

```
  </system.web>
</configuration>
```

ASP.NET maintains cookieless session state by automatically inserting a unique session ID into the page's URL. For example, the following URL has been modified by ASP.NET to include the unique session ID lit3py55t21z5v55vlm25s55:

```
http://www.example.com/(S(lit3py55t21z5v55vlm25s55))/orderform.aspx
```

When ASP.NET sends a page to the browser, it modifies any links in the page that use an application-relative path by embedding a session ID value in the links. (Links with absolute paths are not modified.) Session state is maintained as long as the user clicks links that have been modified in this manner. However, if the client rewrites a URL that is supplied by the application, ASP.NET may not be able to resolve the session ID and associate the request with an existing session. In that case, a new session is started for the request.

The session ID is embedded in the URL after the slash that follows the application name and before any remaining file or virtual directory identifier. This enables ASP.NET to resolve the application name before involving the SessionStateModule in the request.

Note

To improve the security of your application, you should allow users to log out of your application, at which point the application should call the Abandon method. This reduces the potential for a malicious user to get the unique identifier in the URL and use it to retrieve private user data stored in the session.

(ii) Regenerating Expired Session Identifiers

By default, the session ID values that are used in cookieless sessions are recycled. That is, if a request is made with a session ID that has expired, a new session is started by using the SessionID value that is supplied with the request. This can result in a session unintentionally being shared when a link that contains a cookieless SessionID value is used by multiple browsers. (This can occur if the link is passed through a search engine, through an e-mail message, or through another program.) You can reduce the chance of session data being shared by configuring the application not to recycle session identifiers. To do this, set the **regenerateExpiredSessionId** attribute of the sessionState configuration element to **true**. This generates a new session ID when a cookieless session request is made with an expired session ID.

Note

If the request that is made with the expired session ID is made by using the HTTP POST method, any posted data will be lost when **regenerateExpiredSessionId** is **true**. This is because ASP.NET performs a redirect to make sure that the browser has the new session identifier in the URL.

(iii) Custom Session Identifiers

You can implement a custom class to supply and validate SessionID values. To do so, create a class that inherits the SessionIDManager class and override the CreateSessionID and Validate methods with your own implementations. For an example, see the example provided for the CreateSessionID method.

You can replace the SessionIDManager class by creating a class that implements the ISessionIDManager interface. For example, you might have a Web application that associates a unique identifier with non-ASP.NET pages (such as HTML pages or images) by using an ISAPI filter. You can implement a custom SessionIDManager class to use this unique identifier with ASP.NET session state. If your custom class supports cookieless session identifiers, you must implement a solution for sending and retrieving session identifiers in the URL.

(k) Session Modes

ASP.NET session state supports several storage options for session variables. Each option is identified as a session-state Modetype. The default behavior is to store session variables in the memory space of the ASP.NET worker process. However, you can also specify that session state should be stored in a separate process, in a SQL Server database, or in a custom data source. If you do not want session state enabled for your application, you can set the session mode to Off.

(l) Session Events

ASP.NET provides two events that help you manage user sessions. The **Session_OnStart** event is raised when a new session starts, and the **Session_OnEnd** event is raised when a session is abandoned or expires. Session events are specified in the Global.asax file for an ASP.NET application.
The **Session_OnEnd** event is not supported if the session Mode property is set to a value other than InProc, which is the default mode.

(m) Configuring Session State

Session state is configured by using the sessionState element of the **system.web** configuration section. You can also configure session state by using the EnableSessionState value in the **@ Page** directive.
The **sessionState** element enables you to specify the following options:

- The mode in which the session will store data.
- The way in which session identifier values are sent between the client and the server.
- The session Timeout value.
- Supporting values that are based on the session Mode setting.

The following example shows a **sessionState** element that configures an application for SQLServer session mode. It sets the Timeout value to 30 minutes, and specifies that session identifiers are stored in the URL.

```
<sessionState mode="SQLServer"
  cookieless="true "
  regenerateExpiredSessionId="true "
  timeout="30"
  sqlConnectionString="Data Source=MySqlServer;Integrated Security=SSPI;"
  stateNetworkTimeout="30"/>
```

You can disable session state for an application by setting the session-state mode to Off. If you want to disable session state for only a particular page of an application, you can set the EnableSessionState value in the **@ Page** directive to **false**. The EnableSessionState value can also be set to **ReadOnly** to provide read-only access to session variables.

(n) Concurrent Requests and Session State

Access to ASP.NET session state is exclusive per session, which means that if two different users make concurrent requests, access to each separate session is granted concurrently. However, if two concurrent requests are made for the same session (by using the same SessionID value), the first request gets exclusive access to the session information. The second request executes only after the first request is finished. (The second session can also get access if the exclusive lock on the information is freed because the first request exceeds the lock time-out.) If the EnableSessionState value in the **@ Page** directive is set to **ReadOnly**, a request for the read-only session information does not result in an exclusive lock on the session data. However, read-only requests for session data might still have to wait for a lock set by a read-write request for session data to clear.

Procedure:

1.Write a web application demonstrating session management using C# in ASP.Net

Conclusion: