UNIT 4
ASP.NET: Working with Data and Security

(12 Hours) — Advanced .NET Technology (CACS408)

Note: This PDF may contain errors — please cross-check important points with your syllabus, prescribed textbook and teacher's notes before the exam.

Note: The syllabus typo "ADOeNET" means ADO.NET. This unit follows classic ASP.NET Web Forms (.aspx pages, web.config, IIS) on the .NET Framework. ASP.NET Core (Units 5–6) handles some topics differently; differences are mentioned where important.

4.1 Web Application Using ASP.NET

ASP.NET is Microsoft's server-side framework for building dynamic web sites, web applications and web services. The browser sends an HTTP request, the server runs code (C# or VB.NET) and returns HTML. An ASP.NET Web Forms page consists of two parts: the **.aspx** file (markup and controls) and the **.aspx.cs code-behind** file (C# logic).

bealink.com

```
<%-- Default.aspx --%>
<%@ Page Language="C#" AutoEventWireup="true"
         CodeBehind="Default.aspx.cs" Inherits="MySite._Default" %>
<html>
<body>
  <form id="form1" runat="server">
    <asp:TextBox ID="txtName" runat="server" />
    <asp:Button ID="btnHello" runat="server" Text="Say Hello"
                OnClick="btnHello_Click" />
    <asp:Label ID="lblMsg" runat="server" />
  </form>
</body>
</html>

// Default.aspx.cs  (code-behind)
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack) { lblMsg.Text = "Welcome"; }
    }
    protected void btnHello_Click(object sender, EventArgs e)
    {
        lblMsg.Text = "Hello, " + txtName.Text;
    }
}
```

Program 4.1: An ASP.NET Web Forms page with code-behind

Typical project items

Item	Purpose
Web.config	XML configuration: connection strings, authentication, session state, compilation.
Global.asax	Application-level events: Application_Start, Session_Start, Application_Error.
App_Data / App_Code	Database files (.mdf) / shared source code compiled automatically.
bin	Compiled assemblies (.dll) used by the site.
Master pages (.master)	Common layout shared by many pages.

4.2 ASP.NET Architecture

When a browser requests an .aspx page, IIS hands the request to the ASP.NET runtime, which creates an **HttpApplication** object, passes the request through **HTTP modules** (authentication, session, caching) and finally to an **HTTP handler** (the Page class) that produces the HTML response. On first request the page is compiled to a class (IL), and later requests reuse the compiled version.

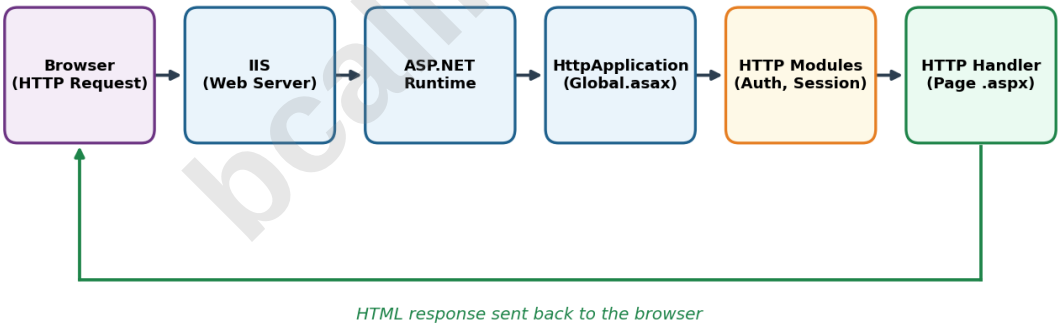


Fig 4.1: ASP.NET request processing pipeline

bcalink.com

Page life cycle

Stage / event	What happens
PreInit	Master page, theme and dynamic controls can be set; IsPostBack is available.
Init / InitComplete	Controls are created and their UniqueID assigned.
Load (Page_Load)	ViewState and postback data are loaded; most code runs here. Use **if (!IsPostBack)** for one-time setup.
Control events	Click, TextChanged, SelectedIndexChanged… handlers execute.
PreRender	Last chance to change controls before they are drawn.
Render	Page and controls produce HTML sent to the browser.
Unload	Cleanup: close connections and release resources.

Note: Easy memory aid: PreInit → Init → Load → Event handling → PreRender → Render → Unload.

4.3 Working with Controls

Control type	Description	Example
HTML controls	Plain HTML tags; client-side only	<input type="text">
HTML server controls	HTML tags with runat="server"; accessible from code	<input id="t1" runat="server">
Web server controls	Rich ASP.NET controls (asp: prefix) with events and state	asp:TextBox, asp:Button
Validation controls	Check user input	RequiredFieldValidator
Data controls	Display/bind data	GridView, DetailsView, Repeater
Navigation controls	Site navigation	Menu, TreeView, SiteMapPath
Login controls	Authentication UI	Login, LoginStatus, CreateUserWizard
User controls (.ascx)	Reusable page fragments you write yourself	Header.ascx

User interface elements

- **Master pages** define a common layout with ContentPlaceHolder areas; content pages fill them.
- **Themes and skins** give a uniform appearance (CSS + control skins).
- **Navigation** through Web.sitemap with Menu, TreeView and SiteMapPath controls.

bcalink.com

- **Standard controls**: Label, TextBox, Button, DropDownList, CheckBoxList, RadioButtonList, Image, Calendar, FileUpload, Panel, MultiView.

4.4 Deployment, Web Sites, Applications and Virtual Directories in IIS

IIS (Internet Information Services) is Microsoft's web server for hosting ASP.NET sites. Deployment means copying the finished site to the server and configuring IIS to serve it.

Term	Meaning
Web site	Top-level IIS container with its own bindings (IP, port, host name) and physical folder. Default Web Site listens on port 80.
Application	A folder marked as an application: runs inside an **application pool**, has its own Global.asax, bin and Web.config.
Virtual directory	A URL path (e.g., /images) mapped to a physical folder anywhere on disk; the folder need not be inside the site folder.
Application pool	Worker process (w3wp.exe) isolating applications; has .NET version and identity settings.

bcalink.com

Deployment methods

- **XCOPY / file copy** of .aspx, bin and Web.config to the IIS folder.
- **Visual Studio Publish** (file system, FTP, Web Deploy) — can pre-compile the site.
- **Web Setup / MSI installer** for installing on multiple servers.
- Steps in IIS Manager: install IIS and ASP.NET → create folder → Add Web Site (or Application / Virtual Directory) → choose application pool and .NET version → set bindings → set folder permissions → browse the site to test.

4.5 Accessing Data Using ADO.NET

Connection strings are stored in Web.config so they can be changed without recompiling. Pages read them through `ConfigurationManager.ConnectionStrings`.

```
<!-- Web.config -->
<connectionStrings>
  <add name="CollegeDB"
       connectionString="Server=.;Database=College;Integrated Security=True"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Web.config: connection string

```
// Default.aspx.cs
using System.Configuration;
using System.Data.SqlClient;

protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack) return;
    string cs = ConfigurationManager.ConnectionStrings["CollegeDB"].ConnectionString;
    using (SqlConnection con = new SqlConnection(cs))
    {
        SqlCommand cmd = new SqlCommand("SELECT Id, Name FROM Student", con);
        con.Open();
        GridView1.DataSource = cmd.ExecuteReader();   // connecting + executing
        GridView1.DataBind();                         // show on the page
    }
}
protected void btnAdd_Click(object sender, EventArgs e)
{
    string cs = ConfigurationManager.ConnectionStrings["CollegeDB"].ConnectionString;
    using (SqlConnection con = new SqlConnection(cs))
    {
        SqlCommand cmd = new SqlCommand("INSERT INTO Student(Name) VALUES(@n)", con);
        cmd.Parameters.AddWithValue("@n", txtName.Text);
        con.Open();
        cmd.ExecuteNonQuery();
    }
}
```

Program 4.2: Connecting to data and executing commands in ASP.NET

Declarative alternative: `SqlDataSource` bound to a `GridView` needs no code. Data-bound controls: GridView, DetailsView, FormView, ListView, Repeater, DropDownList.

bealink.com

4.6 State Management

HTTP is **stateless**: the server forgets the user after every response, and each request creates a new page object. **State management** techniques preserve data between requests.

Client-side (stored in browser / page)	Server-side (stored on server)
ViewState, hidden fields, query string, cookies, control state	Session state, Application state, Cache, database
Less server memory; data visible/tamperable by user	More secure; uses server memory

Page-level state: ViewState

ViewState stores control values in a hidden field (`__VIEWSTATE`) encoded in the page, so values survive postbacks of the *same page*. It can also hold custom values: `ViewState["count"] = 5;`. Disable it with `EnableViewState="false"` to reduce page size.

Using cookies to preserve state

A **cookie** is a small text file (max about 4 KB) stored on the client and sent with each request. A *session cookie* lasts until the browser closes; a *persistent cookie* has an `Expires` date.

```
// Write a persistent cookie
HttpCookie ck = new HttpCookie("UserName", txtName.Text);
ck.Expires = DateTime.Now.AddDays(7);
Response.Cookies.Add(ck);

// Read it
if (Request.Cookies["UserName"] != null)
    lblMsg.Text = "Welcome back, " + Request.Cookies["UserName"].Value;

// Delete: set an expiry in the past
Response.Cookies["UserName"].Expires = DateTime.Now.AddDays(-1);
```

Program 4.3: Cookies

ASP.NET session state

Session state keeps data for one user across many pages. ASP.NET gives each user a unique **SessionID** (sent in a cookie, or in the URL when cookieless) and stores their data on the server. The default timeout is 20 minutes.

```
Session["User"] = txtName.Text;                 // store
string u = (string)Session["User"];             // retrieve (cast needed)
Session.Remove("User");  Session.Clear();  Session.Abandon();   // end session

// Storing an object in session state
[Serializable]                                  // required for out-of-process modes
public class Cart { public int Items; public decimal Total; }

Cart cart = new Cart { Items = 3, Total = 1500m };
Session["Cart"] = cart;
Cart c = (Cart)Session["Cart"];
```

Program 4.4: Session state and storing objects

ocalink.com

Configuring session state (Web.config)

```
<system.web>
  <sessionState mode="InProc" timeout="30" cookieless="false" />
</system.web>
```

Web.config: session configuration

Mode	Where session data is stored	Remarks
InProc (default)	Memory of the web server process	Fastest; lost if the app pool recycles; not for web farms.
StateServer	Separate ASP.NET State Service process	Survives restarts; shared by servers; objects must be serializable.
SQLServer	SQL Server database	Most durable and farm-friendly; slower.

Mode	Where session data is stored	Remarks
Custom	Custom provider (e.g., Redis)	Own storage implementation.
Off	—	Session disabled.

Application state (`Application["hits"]`) is shared by *all* users; use `Application.Lock()` / `UnLock()` when updating it.

4.7 Validation

Validation controls check user input before it is processed. They run on the client (JavaScript, fast feedback) and always again on the server (security). Set `ControlToValidate`, `ErrorMessage` and, in code, test `Page.IsValid` before saving.

Validator	Checks that…
RequiredFieldValidator	the field is not empty.
CompareValidator	value matches another control or a constant (e.g., password confirmation, data type).
RangeValidator	value lies between a minimum and maximum.
RegularExpressionValidator	value matches a pattern (e-mail, phone number).
CustomValidator	your own client/server logic returns true.
ValidationSummary	shows all error messages together.

```
<asp:TextBox ID="txtEmail" runat="server" />
<asp:RequiredFieldValidator runat="server" ControlToValidate="txtEmail"
     ErrorMessage="Email is required" ForeColor="Red" Display="Dynamic" />
<asp:RegularExpressionValidator runat="server" ControlToValidate="txtEmail"
     ErrorMessage="Invalid email"
     ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />
<asp:TextBox ID="txtAge" runat="server" />
<asp:RangeValidator runat="server" ControlToValidate="txtAge" Type="Integer"
     MinimumValue="18" MaximumValue="60" ErrorMessage="Age must be 18-60" />
<asp:Button ID="btnSave" runat="server" Text="Save" OnClick="btnSave_Click" />
<asp:ValidationSummary runat="server" />

// code-behind
protected void btnSave_Click(object sender, EventArgs e)
{
    if (Page.IsValid) { /* save data */ }
}
```

Program 4.5: Validation controls

cairk.com

4.8 Security: Authentication and Authorization

Authentication verifies *who* the user is; **authorization** decides *what* that user may access. ASP.NET authentication modes: **Windows** (uses Windows accounts/domain, good for intranets), **Forms** (own login page and cookie, best for internet sites), **Passport** (obsolete) and **None**.

IIS URL authorization

In Web.config, the `<authorization>` section allows or denies users and roles. Rules are checked top to bottom and the first match wins. Symbols: `*` = all users, `?` = anonymous (not logged-in) users.

```
<system.web>
  <authorization>
    <allow roles="Admin" />
    <allow users="ram,sita" />
    <deny users="?" />        <!-- anonymous users denied -->
  </authorization>
</system.web>

<!-- Protect only one folder or file -->
<location path="Admin">
  <system.web>
    <authorization><allow roles="Admin" /><deny users="*" /></authorization>
  </system.web>
</location>
```

Web.config: URL authorization

bcalink.com

Forms authentication

With **Forms authentication** an unauthenticated user is redirected to a login page. After valid credentials, ASP.NET issues an encrypted **authentication ticket** stored in a cookie (`.ASPXAUTH`) which is sent with later requests.

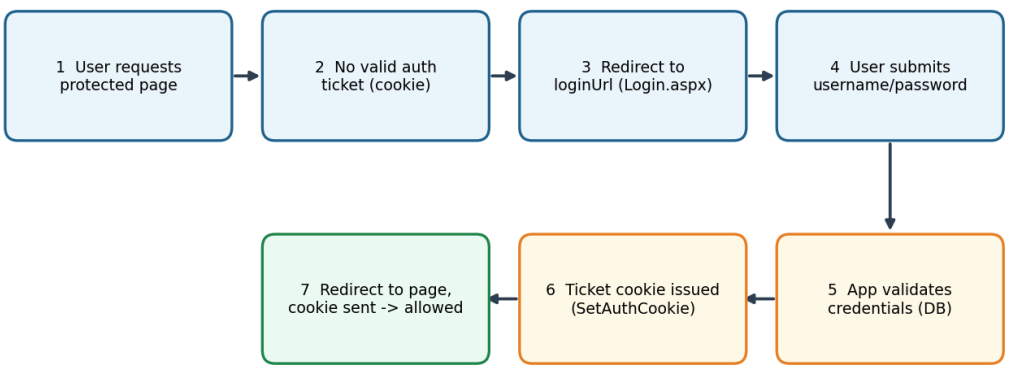


Fig 4.2: Forms authentication flow

```
<system.web>
  <authentication mode="Forms">
    <forms loginUrl="Login.aspx" defaultUrl="Home.aspx" timeout="30"
           protection="All" slidingExpiration="true" />
  </authentication>
  <authorization><deny users="?" /></authorization>
</system.web>

// Login.aspx.cs
protected void btnLogin_Click(object sender, EventArgs e)
{
    if (IsValidUser(txtUser.Text, txtPass.Text))     // check against database
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, chkRemember.Checked);
    else
        lblMsg.Text = "Invalid username or password";
}
// Logout:  FormsAuthentication.SignOut();  Response.Redirect("Login.aspx");
```

Program 4.6: Forms authentication configuration and login

Note: Passwords must be stored as salted hashes, never as plain text, and login pages should use HTTPS.

Config file encryption

Sections of Web.config such as `connectionStrings` can hold secrets. They can be encrypted with the **aspnet_regiis.exe** tool; ASP.NET decrypts them automatically at run time, so no code change is needed.

Provider	Description
DPAPIProtectedConfigurationProvider	Uses Windows Data Protection API; key tied to the machine — simple but not portable between servers.
RsaProtectedConfigurationProvider	Uses RSA key container; keys can be exported — suitable for web farms.

```
aspnet_regiis -pe "connectionStrings" -app "/MySite"          (encrypt)
aspnet_regiis -pd "connectionStrings" -app "/MySite"          (decrypt)
aspnet_regiis -pef "connectionStrings" "C:\inetpub\MySite"  (physical path)
```

Command prompt: encrypting a Web.config section

Note: In ASP.NET Core there is no Web.config-based authentication. It uses middleware (cookie authentication, ASP.NET Core Identity, JWT) with [Authorize] attributes, and secrets are kept in user-secrets, environment variables or Azure Key Vault.

bcalink.com

Key Takeaways — Unit 4

✓ Web Forms page = .aspx markup + code-behind (.aspx.cs); controls with runat="server".

✓ Request path: Browser → IIS → ASP.NET runtime → HttpApplication → Modules → Handler (Page).

✓ Page life cycle: PreInit, Init, Load, events, PreRender, Render, Unload; use IsPostBack.

✓ IIS: Web site > Application (app pool) > Virtual directory; deploy by copy/publish.

✓ ADO.NET in ASP.NET: connection string in Web.config, Command/DataReader, bind to GridView.

✓ State: ViewState (page), Cookies (client), Session (per user), Application (all users); session modes InProc/StateServer/SQLServer.

✓ Validators: Required, Compare, Range, RegularExpression, Custom, Summary; check Page.IsValid.

✓ Security: authorization rules (allow/deny, ? and *), Forms authentication cookie, encrypt Web.config with aspnet_regiis.

— End of Unit 4 —

bcalink.com