Cst556 Distributed Applications for .NET with Mono

6. Web Applications with .NET
6.a Web Applications: Outcomes and References

6.a.1 Motivation and Outcomes

- What are **ASP.NET Web Applications**? Distributed applications in which the client (UI display) is browser rendered and underlying control exists on the server. This section covers construction of client and server-side code.

- **Section Outcomes**
  - To know how to use **ASP.NET user interface controls** to create a browser rendered user-interface for a distributed application.
  - To know how to **develop and deploy** the **C# code-behind** to manipulate a set of **ASP.NET user-interface controls**.
  - To further understand the relationship between user-interface description with **xml/html** based languages (**.aspx/html**) and the underlying code-behind (**C#**) that provides the logic of an application.
  - To understand how to construct and deploy user-defined **ASP.NET user interface controls** (**.ascx**).
  - To understand how to use **ASP.NET** and **Form Authentication** to provide login authentication for a web site or web application.
6.a.2 References and Software for .NET Web Applications

- Necessary software installs for executing examples in this section
  - .NET Software Development Kit or Visual Studio .NET. For the SDK. See: [http://www.microsoft.com/downloads](http://www.microsoft.com/downloads) select product .NET and search for SDK.
  - `webdev.webserver` server or IIS. Preferred is `webdev.webserver` for development purposes. See: [http://cassiniwrapper.codeplex.com/](http://cassiniwrapper.codeplex.com/)

- For on-line reference material, examples of .NET applications and the Framework API see the Microsoft site.
6.a.3 Index to .NET Web Application Example Projects

- A first example: Section 6.b.8 Developing and Deploying the Code-Behind
- File upload: Section 6.b.10 File Upload from an ASPX page
- User defined controls: Section 6.c.2 Example User-Created Control
- Forms authentication: Section 6.d.2 Example Forms Authentication Web Application
6.b Web Applications in C#

6.b.1 .NET Web Applications and Their Context

- **Web Applications** are distributed across the **Browser** and **Web server**
  - Browser renders the GUI for the application (maybe several related pages)
  - Client side may perform basic syntactic checking, such as field validation.
  - The application logic is performed on the server, which may rely on an underlying database or other server technology (RMI or web services) to realize functionality. Communication between client and server is via http and managed by the browser.

- Components of **.NET** Web Applications
  - **ASPX** (active server page extension) - web forms page that specifies the visual components and their connections to the application logic.
  - **ASCX** - user-created visual components (controls) that are managed at the server. Generally used to define components that are shared by multiple web forms pages.
  - **Code-behind** - C# language class for each web forms page or user-created control whose methods are executed on the server to manage web forms controls and implement the application logic.
6.b.2  Web Forms Pages

- Contain visual controls that are defined in markup (xml). The Web server’s container (ASP.NET) converts the markup to html/javascript
- Sample of pre-defined controls that are available in .NET Framework
  - **Label** - text label that can be pre-defined or set by the server code-behind,
  - **Button** - can generate an event in the server code-behind,
  - **DropDownList, CheckBox(List), RadioButton(List)** allows the user to select among alternatives, and can generate an event in the server code-behind that will trigger application specific methods.
  - **TextBox** and **ListBox** - provides single line (lists) of user input or display.
  - **DataGrid and Table** - tabular visual component that can be used for both data display and entry. Can be loaded directly from a database.
6.b.3 Comparing Web Forms Page, Codebehind and Html

• Example of a Button as defined in ASPX file, in corresponding C# code-behind and resulting ASP.NET generated html.
  - ASPX: `<asp:Button id="addButt" runat="server"/>
  - html `<input type="submit" name="addButt" value="Add Post" id="addButt"/>

• Example of TextBox as defined in ASPX file, in corresponding C# code-behind and resulting ASP.NET generated html (Label not included).
  - ASPX: `<asp:TextBox id="name" runat="server"/>
  - C# code-behind: `System.Web.UI.WebControls.TextBox` `name;`
  - html: `<input name="simple:name" type="text" id="simple_name" />`
6.b.4 Processing a Web Forms Page

- The application logic **C# code-behind** class for an **aspx page** is compiled into a dynamic link library (**dll**). Both the **.aspx page** and the **.dll** are made visible to the web server, and associated **ASP.NET** container.

- The first time a user’s browser requests the **aspx page**, ASP.NET generates and compiles a class that represents the page.
  - The class is able to dynamically generate html/javascript that can be returned to the browser to cause proper rendering of the page.
  - The class extends (inherits from) the **C# code-behind** class that provides the application logic.

- User interaction with the Form
  - **Round-trips** are caused by the user filling in a **TextBox** and clicking a **Button**, for example. The click causes **post-back** to the server, execution of a developer specified **C# code-behind** method and sending a new version of the page back to the browser.
  - **Mouse-over** events (such as tool tips) and **form field validation** (such as making sure a text box contains a number) can be handled in the browser and do not require a round-trip.
6.b.5 Developing an ASPX Page

- ASPX file, for example ViewGradeInfo.aspx
  
  - **header**: `<%@ Page language="c#" Codebehind="ViewGradeInfo.cs" AutoEventWireup="false" Inherits="coursetools.ViewGradeInfo" %>
  
  - Mix html with definitions of asp controls to define appropriate layout, such as:
    
    ```html
    <table style="background-color:#FFC20E; ... ">
      <tr>
        <td><b>Enter your name here: </b></td>
        <td><ASP:TextBox id="name" runat="server" /></td>
      </tr>
    </table>
    ```
  
  - All controls with `runat="server"` can be defined and manipulated in the `C# code-behind` class, in this case the class `ViewGradeInfo`.

  - **Visual Studio .NET** tool provides a drag-and-drop interface for developing `aspx` and corresponding `code-behind` files.

  - Class name and namespace are as specified in the header.

  - The ASPX filename is used as the file portion of the URL, such as: `http://pooh.poly.asu.edu/coursetools-gradeswc/ViewGradeInfo.aspx`
6.b.6 Developing the Code-Behind that Corresponds to ASPX

- Notes on the **C# code-behind**
  - Source file, Namespace and Class names match the ASPX header.
  - For a server page, the class extends: *System.Web.UI.Page* such as:
    ```csharp
    public class ViewGradeInfo : System.Web.UI.Page {
    }
    ```
  - Each of the *runat="server"* controls defined on the aspx page may also
    be defined as protected members of the code-behind C# class. They can
    be manipulated as appropriate, for example:
    ```csharp
    protected System.Web.UI.WebControls.TextBox name;
    ...
    String userName = salutation + name.Text;
    ```
  - The class over-rides the **OnInit** method to provide one-time initialization,
    such as defining event handlers. For example, specify the method to
    handle button clicks:
    ```csharp
    override protected void OnInit(EventArgs e)
    {
        InitializeComponent();
        base.OnInit(e);
    }
    private void InitializeComponent()
    {
        this.logoutButt.Click += new System.EventHandler(this.buttonClick);
        this.Load += new System.EventHandler(this.Page_Load);
    }
    ```
6.b.7 Developing the Code-Behind: Page Life-Cycle

- The Life-Cycle for Page objects
  - **OnInit** is called when the page is initialized by the ASP.NET container. This method is used to provide initialization of listeners for events of interest.
  - The **System.Web.UI.Page** class defines a **Load** event that occurs on each page request. **Load** occurs the first time a page is loaded by a client browser and each time a **post back** occurs (e.g., the user clicks a button).
  - Often an application needs to display an initial state for controls (such as show a grid of today’s appointments) when the page is first loaded. This leads to definition of a **Page_Load** method to handle the **Load** event as shown in the **InitializeComponent** method on the previous slide.

```csharp
private void Page_Load(object sender, System.EventArgs e) {
    retrieveUserNPwd();
    if(!IsPostBack){
        getNDisplayAppointmentInfo();
    }
}
```

- In the example above, **retrieveUserNPwd** is performed on every **Load** event (button click for example), and **getNDisplayAppointmentInfo** is performed the first time a client loads the page (get versus post in http).
6.b.8 Developing and Deploying the Code-Behind

- **More on Life-Cycle**
  - Page state and event listener associations persist across client requests
  - Pages are stateless. Code-behind instance variables are not saved between round-trips. Instance variables must be re-initialized for each request.

- **Deployment**, for example a user manager web application (see below).
  - The C# code-behind together with any required user-defined classes must be compiled into a dynamic library (dll).
    
    ```csharp
    csc /t:library /out:c:\deployDir\bin\UserMgr.dll UserMgr.cs
    ```
  - The ASPX (UserMgr.aspx) must be placed in the deployDir
  - IIS or the webdev.webserver must have the deploy directory (deployDir) registered as a virtual directory.

- **Run the example**: usermgr.jar
  - Extract; in the UserMgr directory, do: ant build followed by ant deploy
  - Deploy target runs xsp4. Alternatively, run manually from a console
    ```
    webdev.webserver c:\deploy Dir 2340 /
    ```
    either way you start cassini, it enables the url:
    ```
    ```
6.b.9 Notes on the User Manager Example

- **Life-cycle** revisited. The C# code-behind produced by the application developer manages information via **Page** properties and any background information source such as a database. Two properties within the Page object are useful:

  - **Application** - used to store information that should be available to all components on the page and across all clients that connect to the page. Characterized by a distributed workgroup application such as EMail bulletin board or chat.

  - **Session** - used to store information specific to a single client connection. Characterized by a shopping-cart application. **Session** is also a property of the **Page** class.

- **Application**: This object is used by the UserMgr to store information about users. It looks like a Hashtable to the code-behind. **Application** is a property of the **Page** class (parent of the code-behind class).
6.b.10 File Upload from an ASPX page

- Here is an example demonstrating how to upload a file to an ASP.NET web application.
  - See: fileUpload.jar

- In reading the example note the following:
  - Determining the path separator using System.Environment.OSVersion
  - Dynamically returning html to display a text or image file on the page.
6.c User-Defined Controls for Web Applications

6.c.1 Context and Use

- **Motivation.** A Web application may want some components to be common across all pages in the application. Such as:
  - A user dialogue that may be used on multiple pages of the application,
  - A common look and feel for all pages of the application, such as a Banner,
  - Customized existing **asp** controls with specific characteristics of the application. For example, an application specific **SearchBox** component for obtaining user search string requests. The **SearchBox** may be based on the **asp:TextBox** control with an associated label, Button and icon.

- The structure of a user-created control is similar to a Web Forms Page. There is an **.ascx** script file with a code-behind **C#** class.
  - The **ascx** file may contain **html, user-created controls and asp controls**.
  - The **C# code-behind** class extends **System.Web.UI.UserControl**
  - The **C# code-behind** class must either be in the **ascx** file or available in the same directory since it is compiled each time a client’s browser first references a Web Forms Page that references the user-created control.
6.c.2 Example User-Created Control

- Here is an example web application with 3 user-created controls.
  - Download and extract the jar file: `usercontrol.jar`
  - In the `UserControl` directory, do: `ant build` followed by `ant deploy`
  - Load the page: `http://localhost:2340/FirstTab.aspx`

- Notes to consider when reading the example:
  - Web forms pages are `FirstTab`, `SecondTab`, `ThirdTab`; User-created controls are `Banner`, `SimpleControl` and `TabPanel`.
  - `SimpleControl` and `FirstTab` demonstrate how information can persist round-trips between the browser and server. The `Application` object is set by `SimpleControl` when `Submit` is clicked, and read by `FirstTab` when `What’s My Name` is clicked. `SecondTab` demonstrates `ListBox`.
  - `ascx` files header is of the form:
    ```csharp
    <%@ control language="C#" src="Banner.cs" inherits="Banner"%>
    ```
  - `aspx` files define the control with:
    ```csharp
    <%@ Register TagPrefix="uc2" TagName="Banner" Src="Banner.ascx" %>
    ```
  - `aspx` files use the control with:
    ```csharp
    <uc2:Banner id="myBanner" runat="server"></uc2:Banner>
    ```
6.d Form Authentication for Web Applications

6.d.1 Context and Use

- **Motivation.** Web applications often want to personalize the application to the user or authenticate users. **Forms Authentication** is one mechanism used to accomplish these objectives.

  - **Note:** the approach presented in this section is independent of using SSL (https) for communicating user and password information to the server. .NET passwords are handled with HTML’s **input type=password** which does not echo password text to the display. It also does not encrypt the password before transmitting it to the server.

- **Approach**

  - **Unauthenticated** requests for any pages within and below a virtual directory are redirected to a **Login Page**.

  - When the user provides acceptable authentication information, a **credential cookie** is created and the request is directed back to the originally requested page.

  - Pages, if desired can access the cookie containing credentials to obtain information about the user.
6.d.2 Example Forms Authentication Web Application

- Suppose the UserMgr Web application requires authentication.
  - See: authenticate.jar
  - In the Authenticate directory, do: ant deploy

- Notes to consider when reading the example:
  - The Register page registration TextBoxes use client forms validation.
  - Clicking the Register button causes an email message to be sent (using smtp.asu.edu) to the registration email address notifying a new password.
  - Authentication (handling Sign-In button clicks) SOAP serializes/deserializes user registration information to/from the file IDTable.soap.
  - When a user is authenticated with the Sign-In click, an encrypted authentication cookie is set containing user information.
  - UserMgr obtains user authentication information from the cookie each time the page is loaded. It also removes the cookie when the user logs off.
6.e Master Pages for Web Applications

6.e.1 Context and Use

- See: [http://www.asp.net/master-pages/tutorials](http://www.asp.net/master-pages/tutorials)

- Mechanism to provide a consistent and site-wide page layout.
- Banner, and page bottom with consistent controls on each page
- Change the appearance of the site by changing the master page

- The master page defines mark-up that is common among all content pages.
- The master page may also define regions that are customizable on a content-page by content-page basis
6.e.2 Master Page Layout

- Regions, headers and footers can be defined in a Master Page

Diagram:

- Header Region: Home | Get Started | Learn | Etc.
- Page-Specific Content: (Implemented in the master page as a ContentPlaceHolder control)
- Advertisement
- Recommended Books
- Footer Region: Advertise | About | Privacy
6.e.3 Integration with Content Pages