**WINDOWS PROGRAMMING FINAL STUDY SHEET**

**Security -** Three Categories of Web Security: Content freely available to everyone (public).

Serve the general population but require a login (application-level security, protected).

Intranet sites for a controlled population of users — a company’s employees (private).

**Authentication -** identifies the originator of requests (who).

**Authorization -** defines who can access which pages (what).

ASP.NET supports two forms of authorization:

* + ACL (access control list) authorization, also known as file authorization, based on file system permissions, typically used with Windows authentication.
  + URL authorization, relies on configuration directives in Web.config files, most often used with forms authentication.

**IIS/ASP.NET Server-Side Security Processing -**

IIS Security:

IIS (Internet Information Services) Server

* + a Web server
  + runs in process Inetinfo.exe as SYSTEM
  + accepts connections
  + responds to HTTP requests
* Web applications are deployed in application directories. Remote clients can’t arbitrarily grab files outside application directories.
* IIS assigns every request an access token representing a Windows security principal. The access token enables the operating system to perform ACL checks on resources targeted.
* IIS supports IP address and domain name restrictions.
* IIS supports encrypted HTTP connections using the Secure Sockets Layer (SSL) family of protocols.
* Anonymous access (access by unauthenticated users)
* Request from anonymous users are tagged with IUSR\_machinename’s access token. IUSR\_machinename is an Internet guest account created when IIS is installed, where machinename is usually the Web server’s machine name.

**ASP.NET Security:**

Server Side Processing:

(1) *Client* accesses .ASPX files =>

(2) *Inetinfo.exe (IIS)* generates an access token => *Aspnet\_isapi.dll* sends the request and the token through named pipe or local procedure calls (LPCs) =>

(3) *Aspnet\_wp.exe (ASP.NET)* makes ACL checks on the requested resource and passes access token to the targeted application =>

(4) *Targeted application* uses a HTTP pipeline => HTTP modules => HTTP handlers (mapped in Machine.config).

Two types of access tokens:

* + Authenticated user: authenticated security principal
  + Unauthenticated user: IUSR\_machinename for anonymous login

**Application Security Scenarios -** Three Typical Security Scenarios   
for Web Applications:

* + Pages can be freely browsed by any: no application-level security
  + Intranet application: use Windows authentication and ACL authorization.
  + Internet application with secure page access: use forms authentication and URL authorization.

**Realm -** A realm is a logical security space that encompasses all or part of a web site.

**Database Based Authentication -**

Forms Authentication: Static Structure

* Security settings in an ASP.NET-based web application are configured in the Web.config files.
* The Web.config file in the root directory (which must be an application directory) specifies the authentication mode, application-specific login page.
* The Web.config file in a subdirectory sets the authorization specifics for the directory.
* User credentials can be stored in a database (preferred) or in the root Web.config file.

Forms Authentication : Dynamic Behavior

* The first time a user accesses a protected resource, ASP.NET redirects the user to the login page.
* If the login is successful, ASP.NET then issues the user an authentication ticket in the form of a cookie (cookies need to be enabled by the client) and redirects the user to the page originally requested.
* The ticket allows that user to revisit protected portions without having to login again.
* The ticket’s lifetime can be controlled to determine how long the login is good for.
* ASP.NET creates an authentication cookie, attaches it to the outgoing response, and redirects the user to the page that he or she originally requested. The lifetime of a persistent cookie is independent of the browser session.
* Authorization is applied on a directory-by-directory basis. Web.config files in each directory specify exactly how the files are to be protected.
* ASP.NET checks to see whether a valid authentication cookie is attached to the request. If the cookie exists, ASP.NET extracts identity information. If the cookie doesn’t exist, ASP.NET redirects the request to the login page.
* “Authenticated users” means anyone who has successfully logged in through LoginPage.aspx.
* Valid users are stored in the database.
* The cookie containing the authentication ticket is a session cookie, destroyed when the browser is closed.
* You are not prompted for password again during a session.

**Role Based Authorization -**

Forms Authentication: Role-Based Security

* Use role membership to allow only some authenticated users to view Secret/ProtectedPage.aspx.
* Without roles:

Deny all unauthenticated users.

<deny users="?" />

Deny all users (users=“\*”) except John and Alice.

*<allow users="John, Alice" />*

*<deny users="\*" />*

Allow all except Jeff, Bob, and Mary:

*<deny users="Jeff, Bob, Mary" />*

*<allow users="\*" />*

<allow> and <deny> are order-sensitive.

ASP.NET will stop at <…= “\*”> and ignore any statements that appear after it.

* With roles:
  + “Users” table has a field named “role” that stores each user’s role (group) membership.
  + Grant Developer access to Secret.

*<allow roles="Developer" />*

*<deny users="\*" />*

* + Map the roles to user accounts so that ASP.NET can determine whether the requestor is a developer or not.
  + Place the mapping in the AuthenticateRequest event handler (invoked at the beginning of every request).
  + Can be done in a custom HTTP module or in Global.asax.

Programming Role-based Authentication

Getting Information about Authenticated Users in Your Code

* + ASP.NET stores user information in the HttpContext.User property.
  + Access User through Page.Context.User or simply Page.User, or HttpApplication.User.
  + The “User” property is of the type IPrincipal (an interface defined in System.Security.Principal).
  + Implemented by the WindowsPrincipal class for Windows authentication and GenericPrincipal class for other forms of authentication (along with Windows authentication).
  + GenericPrincipal is a device for representing user identities independent of the authentication protocol being used. ASP.NET compares the role name in the GenericPrincipal to the roles granted access through Web.config.
  + User.Identity contains some usefull properties:

Web Services (Architecture) -

outside

* A Web service is a different kind of Web application.
* It’s not designed to serve end users.
* It’s designed to provide services to other applications through a highly programmable Internet.
* It doesn’t have a user interface.
* It exposes callable API functions, known as Web methods, over the Internet.
* .NET Framework makes writing Web services and Web service clients easy.
* Web services are not the property of Microsoft. They’re an industry standard built on open protocols such as HTTP and the Simple Object Access Protocol (SOAP).
* You don’t need the .NET Framework to write Web services or Web service clients.

Inside

* A Web service is an application that:
  + Runs on a Web server
  + Exposes Web methods to interested callers
  + Listens for HTTP requests representing commands to invoke Web methods
  + Executes Web methods and returns the results
  + Most Web services expect SOAP messages.

Inside ASP.NET based Web services

* ASMX is a file name extension registered to ASP.NET in Machine.config.
* ASMX files begin with @ WebService directives.
* At a minimum, the directive must contain a Class attribute identifying the class that makes up the Web service.
* Web service classes can be attributed with optional WebService attributes.
* Web methods are declared by tagging public methods with WebMethod attributes.
* Helper methods are not exposed by omitting the attribute.
* HTTP, XML, and SOAP are hidden under the hood.
* The Web methods can be invoked with SOAP, HTTP GET, and HTTP POST.
* They can return output in SOAP responses or simple XML wrappers.

**Web Methods -**The WebMethod attribute tags a method as a Web method and supports the following parameters:

|  |  |
| --- | --- |
| Parameter Name | Description |
| BufferResponse | Enables and disables response buffering |
| CacheDuration | Caches responses, in seconds |
| Description | Adds a textual description to a Web method |
| EnableSession | Enables and disables session state for this Web method (default: disabled) |
| MessageName | Specifies the Web method’s name |
| TransactionOption | Specifies the transactional behavior of a Web method |

[WebMethod (EnableSession="true", Description="Adds an item to a shopping cart")]

**WSDL -**

WSDL (Web Service Definition Language) for services

“WSDL (Web Services Description Language) is an XML-based language for describing Web services and how to access them.”

<http://www.w3schools.com/wsdl/default.asp>

The Web Services Description Language (WSDL)

* A new standard for describing web services.
* An XML vocabulary.
* For machines to read.
* Documented at <http://www.w3.org/TR/wsdl>.
* Need to publish a WSDL contract when publishing a Web service.
* Other developers can use the contract to write clients for your Web service. Wsdl.exe generates a wrapper class containing all the elements needed to talk to a Web service.
* To generate a WSDL contract :

[*http://winserv1.cs.uakron.edu/xiaotest/Calc.asmx?wsdl*](http://winserv1.cs.uakron.edu/xiaotest/Calc.asmx?wsdl)

WSDL Contract

* A WSDL contract which contains:
  + service element that describes the Web service;
  + operation elements that document the “operations,” or Web methods, that the service supports;
  + binding elements that document the protocols that the Web methods support;
  + other descriptive information.

SOAP -SOAP (Simple Object Access Protocol) for objects

“SOAP is a simple XML-based protocol to let applications exchange information over HTTP.” <http://www.w3schools.com/soap>

<http://www.w3.org/TR/SOAP>

XML -

**UDDI -**

UDDI(Universal Description, Discovery, and Integration) is a global (Internet) Web service directory that is itself implemented as a Web service.

* + Developed by IBM, Microsoft, and Ariba.
  + A specification for building distributed databases that enable interested parties to “discover” each other’s Web services.
  + No one company owns the databases; anyone is free to publish a UDDI-based business registry.
  + Operator sites have already been established by IBM and Microsoft.
* UDDI sites are themselves Web service sites.
* They publish a pair of SOAP-based APIs:

an inquiry API for inquiring about companies and their Web services

a publisher API for advertising a company’s Web services (operator sites typically limit the publisher API to registered members).

* Most developers will use high-level tools to query UDDI business registries and generate wrapper classes that allow them to place calls to the Web services.

**Web Service Clients (Stand Alone, Web Based) -**

**Web Service Proxies -**

**For-Fee Web Services -**

**Cloud Computing: IaaS -**

Infrastructure as a Service (IaaS)

¢Amazon’s Elastic Compute Cloud, Microsoft Azure VM role instance

**Cloud Computing:PaaS -**Platform as a Service (PaaS)

¢Microsoft Azure, Google App Engine

**Cloud Computing:SaaS -**

Software as a Service (**SaaS**)

¢Web services, Google Apps

**ADO.NET -**

ADO.NETis the database API for managed applications (application servers) to talk to database servers (DBMS: Database Management Systems).

* a database API for managed applications;
* a set of classes in .NET FCL System.Data namespace;
* designed to work over the Web;
* integrates effortlessly with XML;
* maps very well to stateless, text-based protocol HTTP;
* accesses databases through modules known as data providers ( a set of APIs that make the accesses easy to program).

Data Providers (SQL Server .NET Provider, OLE DB .NET Provider) -

**Database Access Patterns -**

Pattern of database programming

* Create a connection object.
* Open the connection.
* Create a command object.
* Execute the command.
* Access the data.
* Close the connection.

**Connections -**

Connection objects represent physical connections to a database.

*SqlConnection or OleDbConnection*

**Commands -**

Command objects represent the commands performed on a database.

*SqlCommand or OleDbCommand*

**DataReaders -**

DataReader objects represent the data obtained by the commands.

Reads data.

* Reads **schema** (meta data) .
* Stream-based access to the results of database queries.
* Fast and efficient.
* Read-only and forward-only.
* Closing a DataReader: reader.Close( ) does NOT close the connection, only frees it for others to use.
* D-E-F-E-N-S-I-V-E P-R-O-G-R-A-M-M-I-N-G.

**DataSets -**

**Set-based Database Accesses**

* capture an entire query in memory
* support backward and forward traversal
* edit data and propagate the changes back to the database.

**DataAdapters -** used to input your sql query

**DataGrid**

* DataGrid is an ASP control for displaying datasets.
* Database displaying procedure:
  + Use DataAdapter to get data from the database.
  + Fill the data into a DataSet
  + Bind the DataSet to a DataGrid
  + Select the fields (columns) to be displayed and their header texts.

**Transactions -**

* A transaction is a logical unit of operations grouped together.
* If one of the operations fails, the others will fail (or be rolled back).
* Distributed transactions — transactions that span two or more databases.
* The .NET Framework supports distributed transactions.
* The .NET supports local transactions (one database):

**XML -**

* Extensible Markup Language.
* De facto data language. http://www.w3.org/TR/REC-XML.
* HTML expresses appearance; **XML describes data and its structure**.
* Text based (platform-independent).
* Object-oriented data representation.
* Has no predefined tags.
* Provides rules to format data.
* Many XML parsers already available.
* Strong XML support in the FCL.

**XSD -**

* XML Schema Definitions (XSDs). http://www.w3.org/TR/xmlschema-1 http://www.w3.org/TR/xmlschema-2.
* Schema: a collection of meta data.
* Meta data: data that describes data.
* XSD is an XML-based language for describing XML documents and the types that they contain.

**DOM -**

* Most XML parsers implement one of two popular APIs: DOM or SAX.
* DOM: Document Object Model http://www.w3.org/TR/DOM-Level-2-Core

**SAX -**

* SAX: Simple API for XML, unofficial, http://www.saxproject.org.

**XPath -**

* XML Path Language
* For addressing parts of an XML document.
* “/Guitars/Guitar” is an XPath expression.
* <http://www.w3.org/TR/xpath>.

**XSL -**

* XSL is a language for expressing style sheets.
* Adopted from [CSS](http://www.w3.org/Style/CSS), a file that describes how to display an XML document of a given type.
* Styling requires a source XML documents, containing the information that the style sheet will display and the style sheet itself which describes how to display a document of a given type. It supports: **Formatting Objects.**
* It also adds a transformation language for XML documents: **XSLT**.

**XSLT -**

* Extensible Stylesheet Language Transformations.
* Converting XML documents into HTML documents.
* Converting HTML documents into other XML documents.

**.NET: WCF -** Windows Communication Foundation

* For distributed applications.
* Using service oriented architecture (SOA).
* Clients can consume multiple services; Services can be consumed by multiple clients. (M:M)
* Services have WSDL interface.
* WCF examples: WSS (Web Services Security, extension to SOAP to apply security to web services), WS-Discovery (Web Services Dynamic Discovery, a multicast discovery protocol to locate services),
* **Endpoints**: client connects to a WCF service at an Endpoint, each service exposes its contract via endpoints.
* End point **ABC**: address, binding, contract
* WCF endpoints use SOAP envelope to communicate with clients (for platform independence).
* **Behaviors** allow the developer to customize how the messages are handled.

**.NET:WPF -** Windows Presentation Foundation

* Graphical subsystem.
* Based on DirectX
* 2D and 3D graphics, vector graphics and animation
* Remote or standalone
* Safe remote view with IE.
* Uses XAML to define UI elements.
* XAML: eXtensible Application Markup Language

**.NET:WF -** Windows Workflow Foundation

Workflow: a series of distinct programming steps.

* An activity at each step.
* Workflow Designer in Visual Studio.
* Workflow engine: scheduling, managing, tracking workflows.
* To create applications that execute an ordered business process (UA curriculum proposal approval system).

**.NET:MVC -**

**.NET:LINQ -** Language Integrated Query

var results = **from** c in SomeCollection

**where** c.SomeProperty < 10

**select** new {c.SomeProperty};

**.NET:Task Parallel -**

Managed concurrency library

* TPL: Task Parallel Library
* PLINQ: Parallel LINQ
* Multithreading based.
* Take advantages of muti-core (Intel) and many core (Nvidia GPU)

**.NET:IoT -**

**.NET -** framework for developing an OS platform independent Programming language independent, web enabled and distributed applications

**FCL -** (.NET framework class library)object-oriented API for writing managed applications more than 7,000 classes in named spaces: e.g. System.Windows.Forms, System.IO stored as DLLs (Dynamically Linked Libraries).

**CIL -**  Common Intermediate Language

**CLR -**  Common Language Runtime defines CTS. makes.NET platform independent defined as runtime environment, the execution engine for managed applications.

**CTS -** Common Type System

**managed code -** does not support multiple inheritance, application code whose every action is subject to approval by the CLR (common language runtime) and its compiler conforms to the CLS (common language specification) and supports the CTS (common type system). Advantages of running managed code through CLR:

Type safe (type checked by CLR at runtime) Light weight Garbage collection

**garbage collection -**

**managed module -**

**contents of a managed module -** CIL instructions generated from the source code Metadata describing code inside the module A CLR header containing important information about the module A Windows Portable Executable (PE) file header

**Metadata -**  a collection of tables that describe the code. TypeDef Class Names

**Reflection -** runtime understanding of the code (data types, program organizations, …)

**Assembly -** is a collection of one or more files (modules) grouped together to form a logical unit.

**Composition of an assembly -**  Manifest: metadata for an assembly. Name, version, data types exported

**JIT -** Just in time compiler, compiles the complete code all at once into native binary code before execution

**C# class -**  group of code and data to be instantiated to form objects. Four categories of class members: Fields: member variables Methods: member functionsProperties: fields exposed using accessor (get and set) methodsEvents: notifications a class is capable of firing

**Data types -** describe the memory layout of objects. The name of an object is the name of the memory space stores its data value

**Struct -** user-defined value types, less overhead and easier to use than classes. The compiler uses “new” to initialize an object.

**Interface -** An interface is a group of zero or more abstract methods Abstract methods have no default implementation.Abstract methods are to be implemented in a child class or child struct. An interface defines a contract between a type and users of that type. Used to define software interface standards.

**abstract class -**An abstract class is a class that can’t be instantiated, i.e., one can’t use an abstract class to create an object.The definition of an abstract class looks like a regular class except the preceding keyword “abstract”. It can have member fields and methods.

**Field -** member variables

**Method -**A method is a (function) pointer that points to the code location of the method in the “text” memory, i.e., the pointer stores the address of the code location of the method in the “text” memory. - member functions

**Property -** A new construct in C# named “Property” makes member fields secure and easy to access at the same time.

**Stack object -**In CTS, Value Types are Stack Objects: memory allocated at compile time on the stack auto destruction, no garbage collection needed less overhead, code runs faster

**Heap object -** In CTS, Reference Types are Heap Objects: memory allocated at run time on the heap garbage collected more flexible, sizes need not to be known at compile time more overhead, code runs slower

**Value type -** int i;“i” is the name of a memory space that stores an integer value. - Struct defines value types (stack objects), even though “new” is used to create struct objects. Value types can’t derive from other types except interfaces

**Reference type** - Class defines reference types (heap objects)

**Boxing -** creates a copy of a value type on the managed heap (converts from value type to reference type)

**Unboxing -**duplicates a reference type on the stack (converts from reference type to value type)

**Exception handling -** (object-oriented event-driven runtime error handling) The CLR defines how exceptions are thrown and how they’re handled. (How exception “events” are generated and how they’re handled by exception “event” handlers).

**GUI -** Graphical User Interface, to make programs easy to use, WYSIWYG (What you see is what you get).

**EDP -**Two main tasks of GUI-based EDP: Design interface appearance (look) Implement even handlers (feel). Event-Driven Programming (EDP): Application waits (idles) after initialization until the user generates an event through an input device (keyboard, mouse, …). The OS dispatches the event to the application who owns the active window. The corresponding event handler(s) of the application is invoked to process the event.

**Windows form/control -** for writing GUI based applications (Windows Applications).

**IP -**  (Internet Protocol) => “Binary” (low-level Internet transmission protocol)

**HTTP -** (HyperText Transport Protocol) => “Intermediate Language” (high-level Internet transmission protocol)

**HTML -** (Hyper Text Markup Language)=> High Level Language (for writing web-pages in)

**HTML form -** input or output in a form tag in html client side

**CGI -**Common Gateway Interface (CGI)

* CGI applications write HTTP responses to standard output (stdout) on the server, which are then forwarded to the client browser by the web server.
* CGI defines a low-level programmatic interface between Web servers and applications that run on Web servers.
* CGI applications can be written in any programming language.
* CGI applications read the input accompanying postbacks through server environment variables and standard input (stdin).
* Slow, restarts a process on every request.

**ISAPI -** Internet Server Application Programming Interface. ISAPI extensions are Windows DLLs hosted by IIS. They’re referenced by URL just like HTML files. IIS forwards HTTP requests to an ISAPI DLL by calling a special function exported from the DLL. The DLL generates HTTP responses. Faster than CGI (run in the same process as IIS). Once loaded, they remain in memory. They’re difficult to write.

**IIS -** Internet Information Services

**ASP.NET -**ASP based on the .NET framework, one of the most popular web programming techniques. Code on the server that dynamically generates HTML for the clients at runtime.

**ASP.NET form -** Web Forms are GUI-based EDP web pages built around controls and event handlers. .NET web forms are processed on the server side.Web forms use HTML, HTTP and IP to transmit and display GUI into a client web-browser.

**ASP.NET control -** Two types: Web Controls and HTML Controls Web Controls Class names are prefixed with asp:. Classes are from System.Web.UI.WebControls. The name of the object is defined by the ID attribute. ASP Web controls are rendered into HTML. ASP Web controls are highly programmable. They support methods, properties, events. <asp:TextBox Text="2" ID="op1" runat="server" /> This web control initializes the textbox to display 2. Any public property of a control can be used this way. Control properties can be accessed from server-side scripts (code between the <script> and </script> tags). Event-driven programming. Controls fire events when users click on them. Wiring an event to an event handler is accomplished by prefixing the event name with “On”. <asp:Button Text=" = " RunAt="server" OnClick="OnAdd" /

**Web Forms programming model -** Three principles of the Web Forms programming model: A Web form’s user interface is “declared” using a combination of HTML and server controls. Server controls fire events that can be handled by server-side scripts. Server-side scripts in ASP.NET are compiled to CIL and executed by the CLR on the server. RunAt=“server” must be used in every tag that ASP.NET is to process

**page-level event -** As the Page object executes, it fires a series of events (page-level events) that can be processed by server-side scripts:

* 1. “Init”, which is fired when the page is first instantiated, and
  2. “Load”, which is fired after the page’s controls are initialized but before the page renders any output.

**code behind -**ASPX files are in text files, anyone can read it. When a company sells its ASPX server programs, it does not want people to see their source code. How? Code-behind is designed to protect the source code. For the static code in HTML, we can’t do anything about it. Dynamic code in C# or other .NET languages can be separated out and compiled into DLLs. Only the static HTML code and the DLLs are delivered to the customs.The dll can be written in any server-side compiled programming languages.

**User control -**

**Custom control -**

**enterprise application -** User Interface, Data Storage, Security, Business Logic, Networking. To deal with those issues effectively, modern enterprise applications are usually designed as multi-tier applications.

**Multi-tier application -** Client: interface to the user. It should be as thin as possible. Thin-client: no software to install on the client site except a standard web browser. Thin-client makes applications easy to deploy, easy to maintain and easy to upgrade. Web Server: communicates with the user interface. Application Server: business logic implemented here with tools from ASP.NET, J2EE, WebLogic (BEAS), WebSphere (IBM). Most application servers have separate modules to dynamically generate user interfaces to be sent to the client by the web server. Database Server: persistent data stored for the application. All three server could reside in the same server hardware.

**Web application -** are applications built for the Web using the .NET framework. The applications use Web forms to provide user interface, per-user data stores to hold shopping carts, caching services to boost performance, and security services to identify users and prevent unauthorized accesses.

.NET Web Applications are actually Application Servers in the four-tier architecture. Other programs (client, web server, database server) need to be there to make the applications work. The those programs can be shared with other applications.

**XCopy -** The Web.config File

To support XCOPY installs— to install applications by copying them to a directory and uninstall them by deleting the files and directories.

**Web application configuration -** Strings defined in the .config file can be retrieved in the program at run time:Web.config is the XML file in which ASP.NET applications store configuration data.

• Not in the registry anymore.

• Case sensitive.

• Inherited, can be overridden by subdirectories.

• Machine.config is at the root. Under

**application cache -**

to improve application performance

• data stored in memory

• as dictionaries of key/value pairs

• string keys

• available to all parts of an application (global)

• Application Cache is replacing Application State

**Application state -**

**Session state -** like global variables

**reflections:**

Assembly a = Assembly.Load (name);

Type type = a.GetType ("PlugIn");

Object obj = Activator.CreateInstance (type);

MethodInfo method = type.GetMethod ("GetImage");

Image image = (Image) method.Invoke (obj, null);

**Event handlers:**

//Create and initialize the button

Button MyButton = new Button ();

MyButton.Location = new Point (16, 16);

MyButton.Size = new Size (96, 24);

MyButton.Text = "Click Me";

// add the button to the form’s Controls collection.

Controls.Add (MyButton);

// Add event handlers to events

MyButton.Click += new EventHandler (OnButtonClicked);

// Write the event handlers

void OnButtonClicked (Object sender, EventArgs e){… }

**Key Components of EDP Event-Driven Programming:**

(1) Event Generators:

keyboard, GUI items (buttons, menus, …), NUI devices.

used by programmers / GUI designers.

(2) Events / Messages:

MouseClick, KeyDown, …

(3) Event Loop:

an infinite loop constantly waits for events.

(4) Event Mapping / Event Registration:

inform event dispatcher which event an event hander is for.

- to be used by the programmers

with appropriate matching:

event – delegate – even handler

(5) Event Dispatcher:

dispatch events to the corresponding event handlers.

(6) Event Handlers:

methods for processing events.

OnMouseClick(), …

to be implemented by programmers.

**Structure of an ASP.NET Web Application:**

An ASP.NET (web) application (server) consists of all the files in a virtual directory and its subdirectories on the HW server.

* ASPX files containing Web forms (unlimited)
* ASCX files containing user controls (unlimited)
* Web.config files containing configuration settings (one per directory)
* A Global.asax file containing global application elements (only one for the entire application)
* DLLs containing custom types employed by the application (unlimited, must be in the bin directory under the root of the virtual directory)

DB example

*void Page\_Load (Object sender, EventArgs e)*

*{*

*if (!IsPostBack) {*

*string ConnectString =*

*ConfigurationSettings.AppSettings["connectString"];*

*MySqlDataAdapter adapter = new MySqlDataAdapter*

*("select \* from titles where price != 0", ConnectString);*

*DataSet ds = new DataSet ();*

*adapter.Fill (ds);*

*MyDataGrid.DataSource = ds;*

*MyDataGrid.DataBind ();//Bind data to GUI*

*}*

*}*