Section 5 Class Notes

Website is a collection of files on disk, and is not like an application. Web applications are designed to contain code and perform logic.

Within a web app, you have a lot of different files dealing with different responsibilities. Models folder, Controllers Folder, etc. All code at compile time gets created into a single assembly. That is very different from how websites work.

A package is a set of related binaries. How we pull in 3rd party dependencies that aren’t part of the framework. We rely on a lot of 3rd party dependencies for web apps.

NuGet is the package manager .NET uses. THE package manager for .NET. Has thousands of 3rd party packages.

URL

{Scheme}://{host}[:port]/{path}[?querystring]

Schema – http, https

Host – also known as domain. series of names separated by dots. Everything between a dot is a domain or subdomain.

Path – specifying which item in host you want

Querystring – same as parameters and functions. Always in format of key/value pairs.

In URLs, you have to use escapes for special characters. For example, since & represents a querystring, you have to use the hexadecimal representation of an & in the url. &amp; => &.

The above is an absolute URL. This is what the web server needs. However, you have to use Relative URLS’s because of moving the app to different servers, etc. (look this up)

Relative URL’s go by slashes. The / is the root (starting point of the app). It takes up the Scheme/host/and port.

Virtual Paths is what we will use in .Net a Virtual Path starts with a ~. ~/path. You will never see that in the browser. It is strictly a server side implementation. The server handles the mapping for you.

Web Forms

MVC

Web Api / REST

**Web Forms** – originally developed when .NET was introduced. Designed to mimick WinForms. WYSIWYG. Drag/Drop controls. Set Properties, etc. .aspx or .ascx extension. Behind the scenes, they are html files with server code mixed in with it. Generates a code-behind file. So, you have form for the designer, and a file for the code. Incredibly heavy. Everything is server-based. So a call to the server was made all the time. Web Forms is a server-side implementation. In order to get Web Forms to work well, you have to pass data back and forth continuously. Very heavy network traffic.

**MVC** – very lightweight. Designed to work with javascript. Not much server-side functionality. Because of that, there is no user-interface for the apps. You have to know HTML.

Model – Data

View – UI

Controller – Glue

Database -🡪 Controller 🡪 UI

Database goes through controller, then sends the data to the UI (view) as a model.

**Web Api / REST** – Can work with either Web Forms or MVC. Basically, it is a set of methods that you can call to allow client to go back and forth between client and server. You don’t go through the server together… that would be expensive.

**Controller** – nothing more than a Type. Each controller is responsible for one and only one resource. Within that resource, you have the standard operations for a resource. A Get, Get All, Add, Update, etc.

**Convention over Configuration** – rather than you configuring every little detail… you are going to rely on convention that has been built-in.

Controllers must adhere to the following Conventions:

* Must derive from type Controller
* Must have a suffix of Controller
* Must be public and not abstract
* Reside in Controllers folder

Controller names are singular. Namespace rules do not apply to Controllers. You CANNOT use a duplicate name for a controller. This even includes controllers that are referenced in your assembly.

Actions are regular methods that must always return either ActionResult or one of the derived types.

Action conventions:

* Public method
* Returns ActionResult

ViewResult is what renders HTML. (lookup how this is different than ActionResult)

ViewName is the name of the actual file on disk.

Why do we need a model?

The layout of the data should be the concern of the data layer. The controller will get the data. What the UI needs may or may not resemble what’s in the database. The model is the flattened down version of your database. It’s sole purpose is to get data from database to view the UI needs. You may have the same model for different views. The model gives the data that the UI needs… and nothing more. Models are just types. Referred to as POCO and DTO.

View

Each view may have one or zero models.