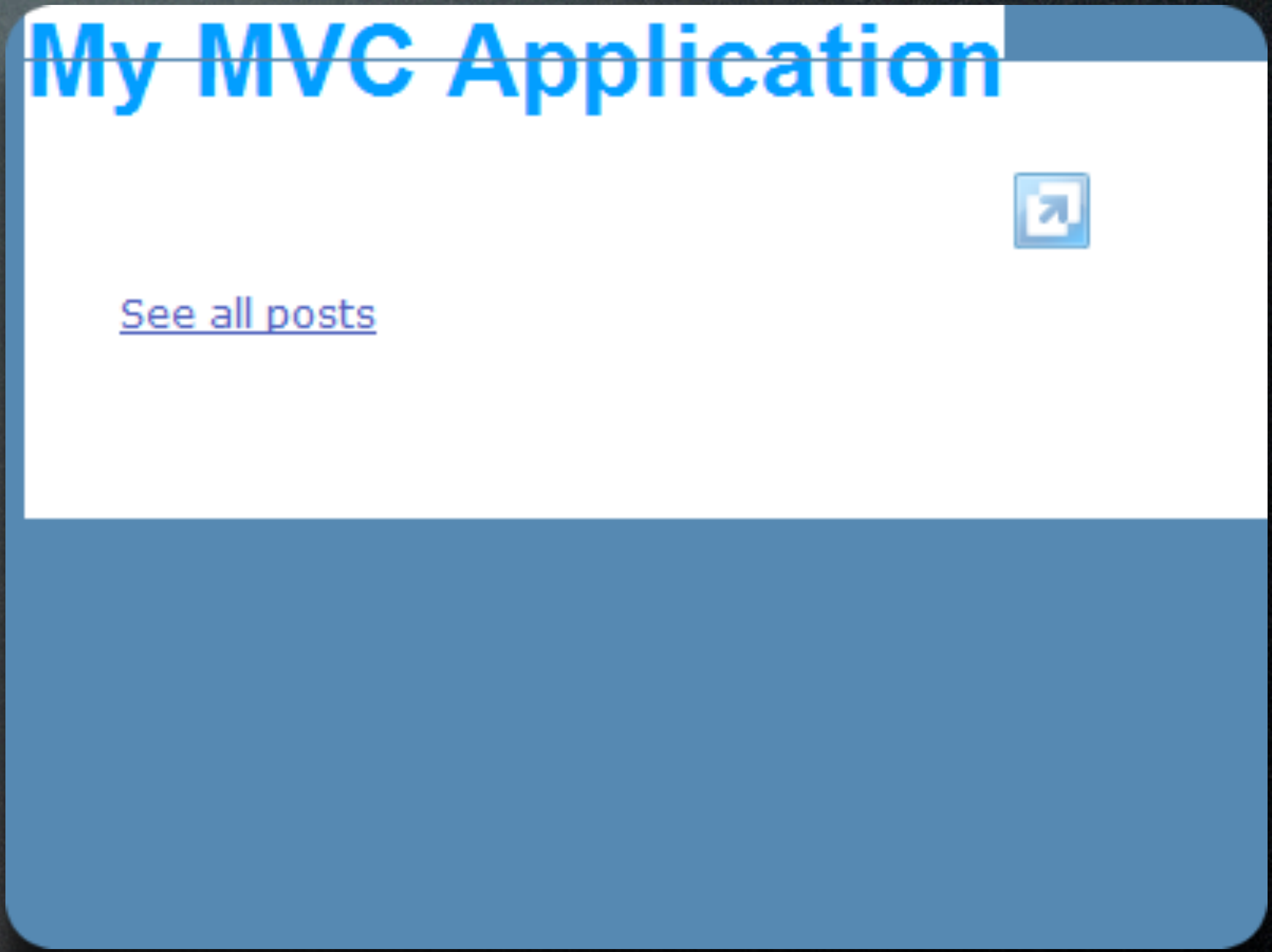
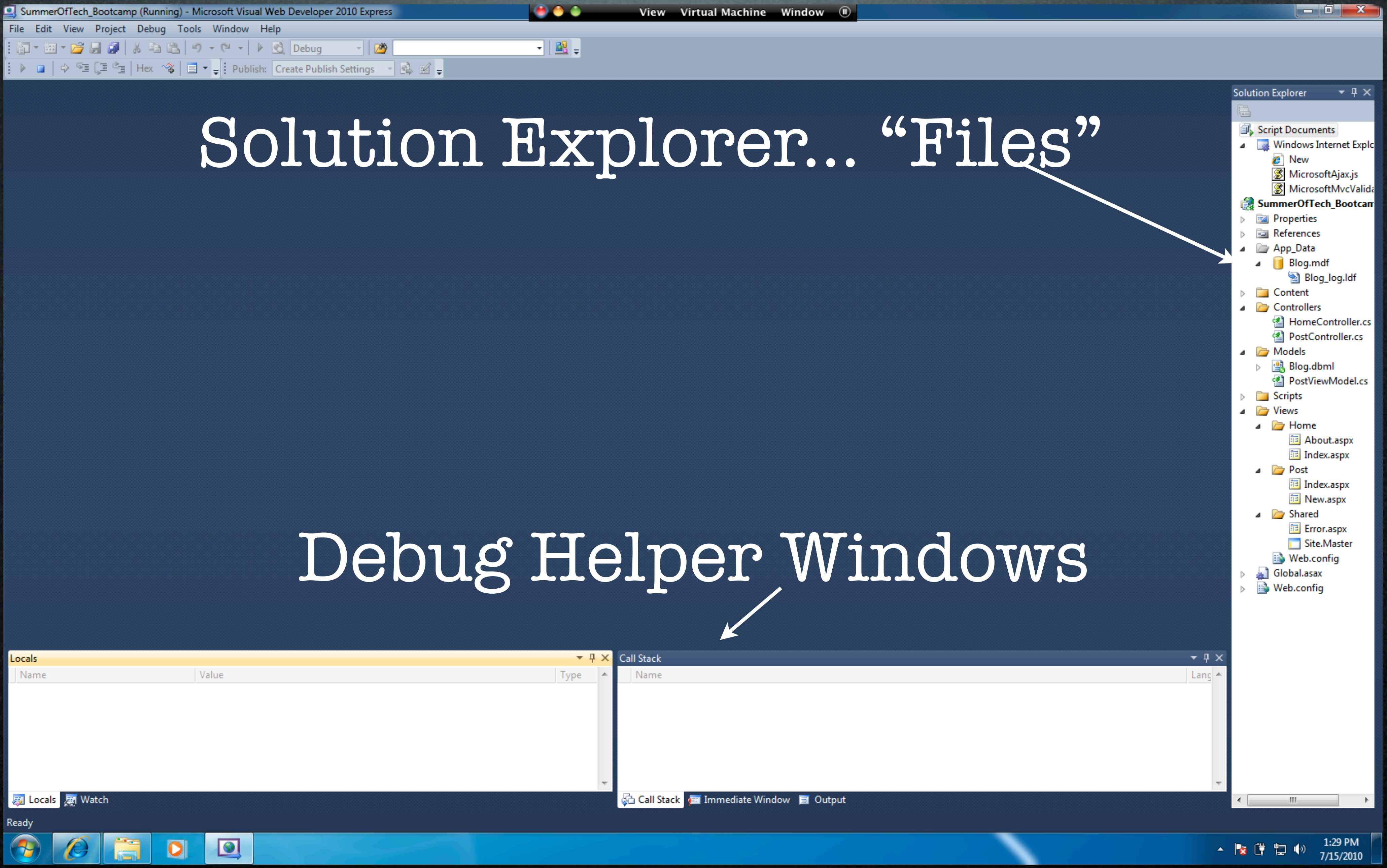
Hands on With ASP.net MVC

YABD- Yet another Blog Demo

Explore the basic project

- Open up Visual Web Developer 2010 Express
- Choose SummerOfTech_Bootcamp in “Recent Projects”
- Hit F5 to run the project
- WOW! not much to look at!
- Click on “See all Posts”
- Click on “Add new Post”
- Click on Create
    - Check out the error messages
- Type in a title and some content
- Click on create
- See it added to the list!



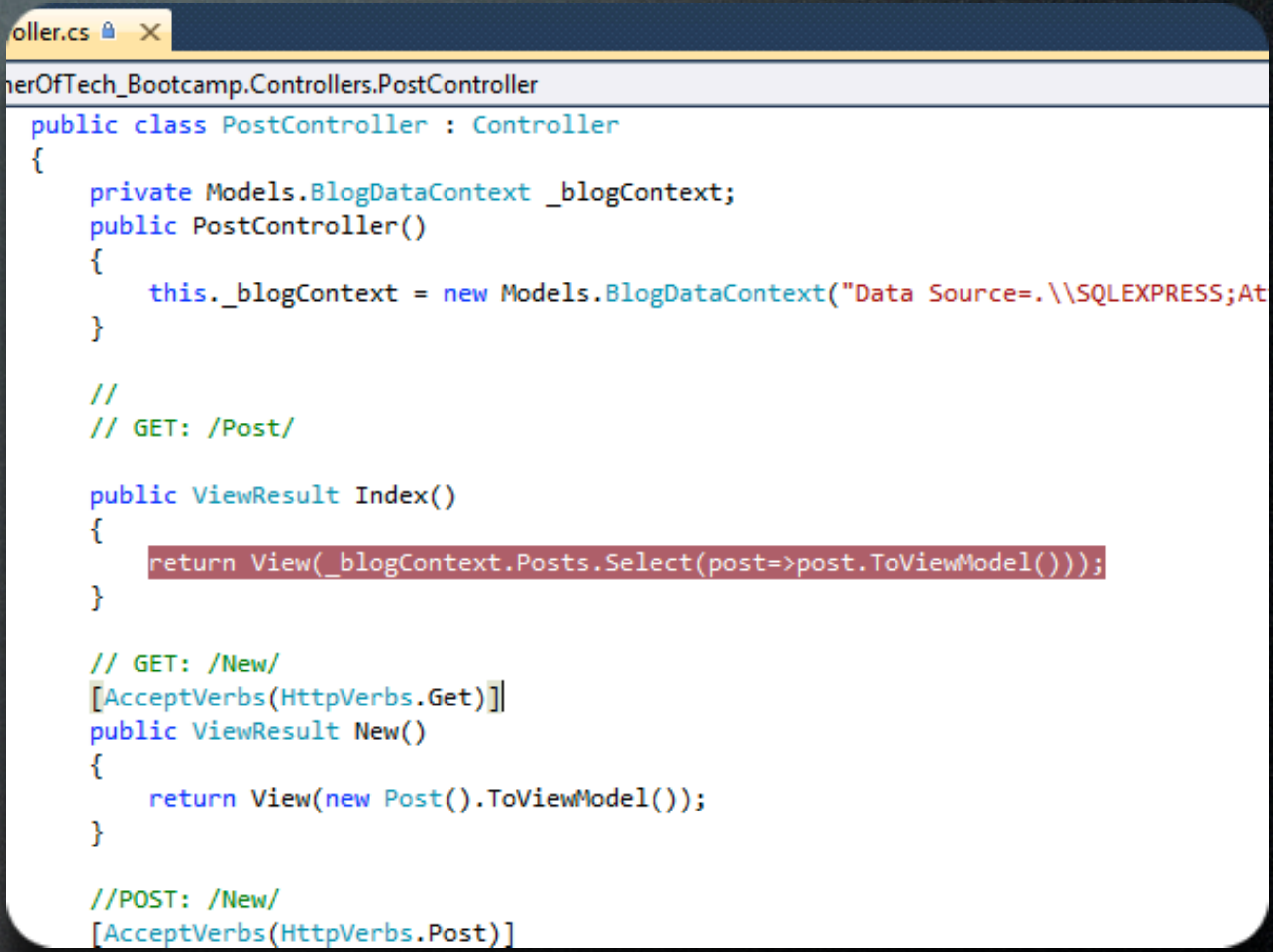

Solution Explorer... "Files"

Debug Helper Windows

Visual Studio Express

Lets debug

- Find “PostController.cs” in “Controllers” folder [in solution explorer] and open it up
- Find Index() method
- click in the margin (grey area) on the one line in this method (should add a red bar)
- now go to the list of posts page and you should see the code halts



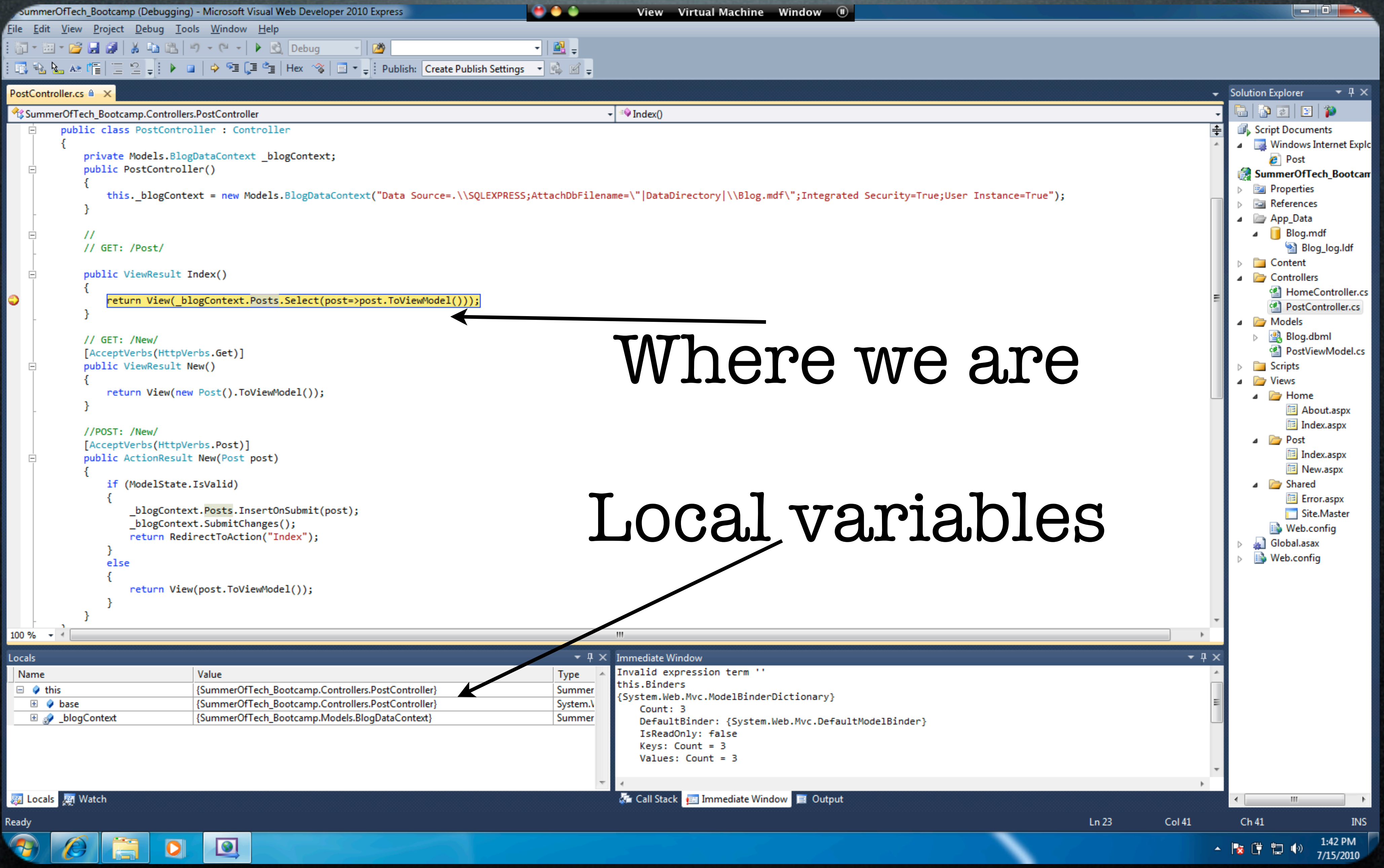
```
oller.cs
nerOfTech_Bootcamp.Controllers.PostController
public class PostController : Controller
{
    private Models.BlogDataContext _blogContext;
    public PostController()
    {
        this._blogContext = new Models.BlogDataContext("Data Source=.\\SQLEXPRESS;At
    }

    //
    // GET: /Post/

    public ViewResult Index()
    {
        return View(_blogContext.Posts.Select(post=>post.ToViewModel()));
    }

    // GET: /New/
    [AcceptVerbs(HttpVerbs.Get)]
    public ViewResult New()
    {
        return View(new Post().ToViewModel());
    }

    //POST: /New/
    [AcceptVerbs(HttpVerbs.Post)]
```


SummerOfTech_Bootcamp (Debugging) - Microsoft Visual Web Developer 2010 Express
File Edit View Project Debug Tools Window Help
PostController.cs
SummerOfTech_Bootcamp.Controllers.PostController
Index()
Solution Explorer
Where we are
Local variables

```
public class PostController : Controller
{
    private Models.BlogDataContext _blogContext;
    public PostController()
    {
        this._blogContext = new Models.BlogDataContext("Data Source=.\\SQLEXPRESS;AttachDbFilename=\"|DataDirectory|\\Blog.mdf\";Integrated Security=True;User Instance=True");
    }

    //
    // GET: /Post/

    public ViewResult Index()
    {
        return View(_blogContext.Posts.Select(post=>post.ToViewModel()));
    }

    // GET: /New/
    [AcceptVerbs(HttpVerbs.Get)]
    public ViewResult New()
    {
        return View(new Post().ToViewModel());
    }

    //POST: /New/
    [AcceptVerbs(HttpVerbs.Post)]
    public ActionResult New(Post post)
    {
        if (ModelState.IsValid)
        {
            _blogContext.Posts.InsertOnSubmit(post);
            _blogContext.SubmitChanges();
            return RedirectToAction("Index");
        }
        else
        {
            return View(post.ToViewModel());
        }
    }
```

Locals

Name	Value	Type
this	{SummerOfTech_Bootcamp.Controllers.PostController}	Summer
base	{SummerOfTech_Bootcamp.Controllers.PostController}	System.\
_blogContext	{SummerOfTech_Bootcamp.Models.BlogDataContext}	Summer

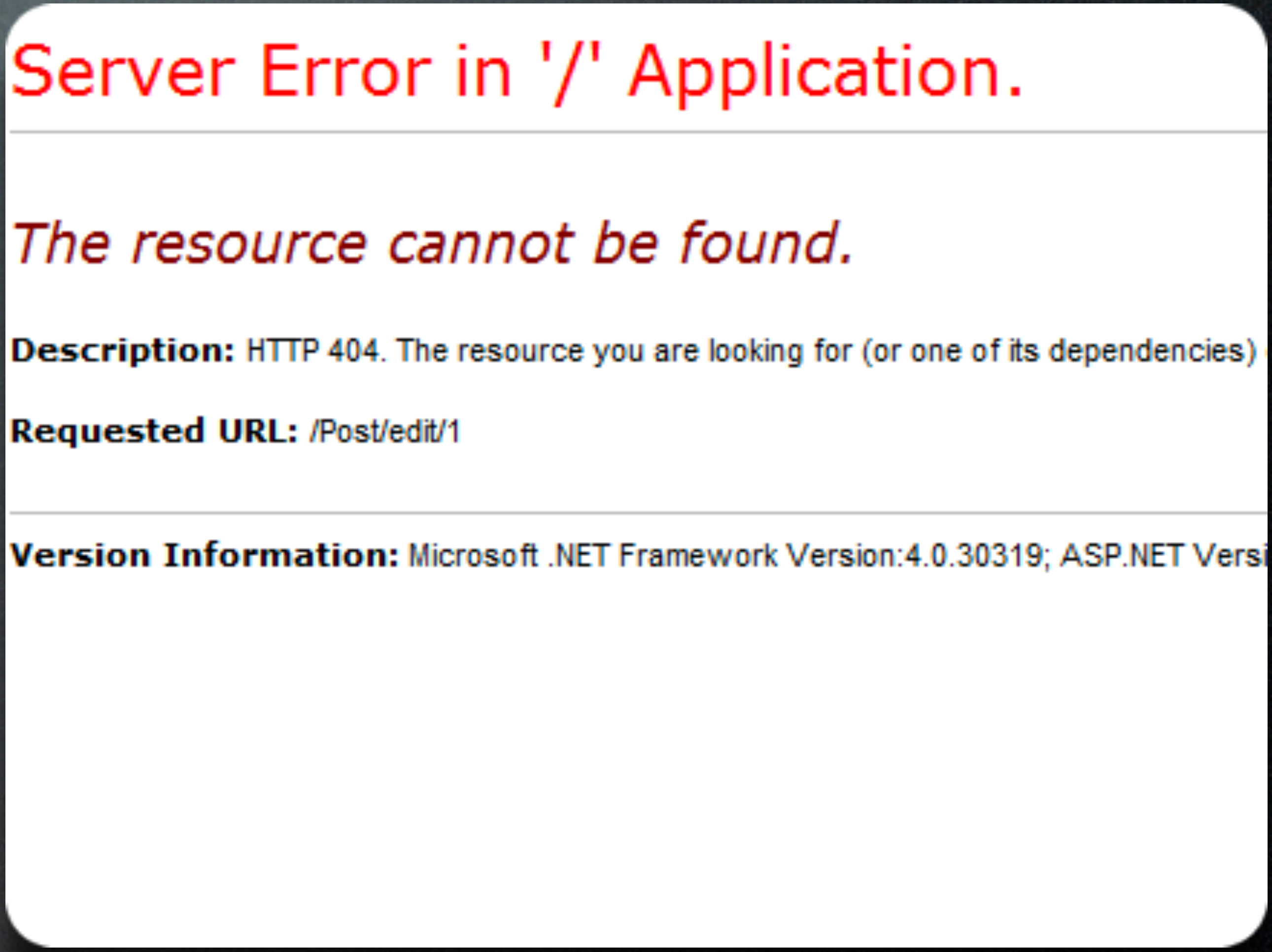
Immediate Window

```
Invalid expression term ''
this.Binders
{System.Web.Mvc.ModelBinderDictionary}
    Count: 3
    DefaultBinder: {System.Web.Mvc.DefaultModelBinder}
    IsReadOnly: false
    Keys: Count = 3
    Values: Count = 3
```

click f5 to continue

Missing code

- try and click on the name of the post you created
- whoops! 404


Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies)

Requested URL: /Post/edit/1

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Versi

why?

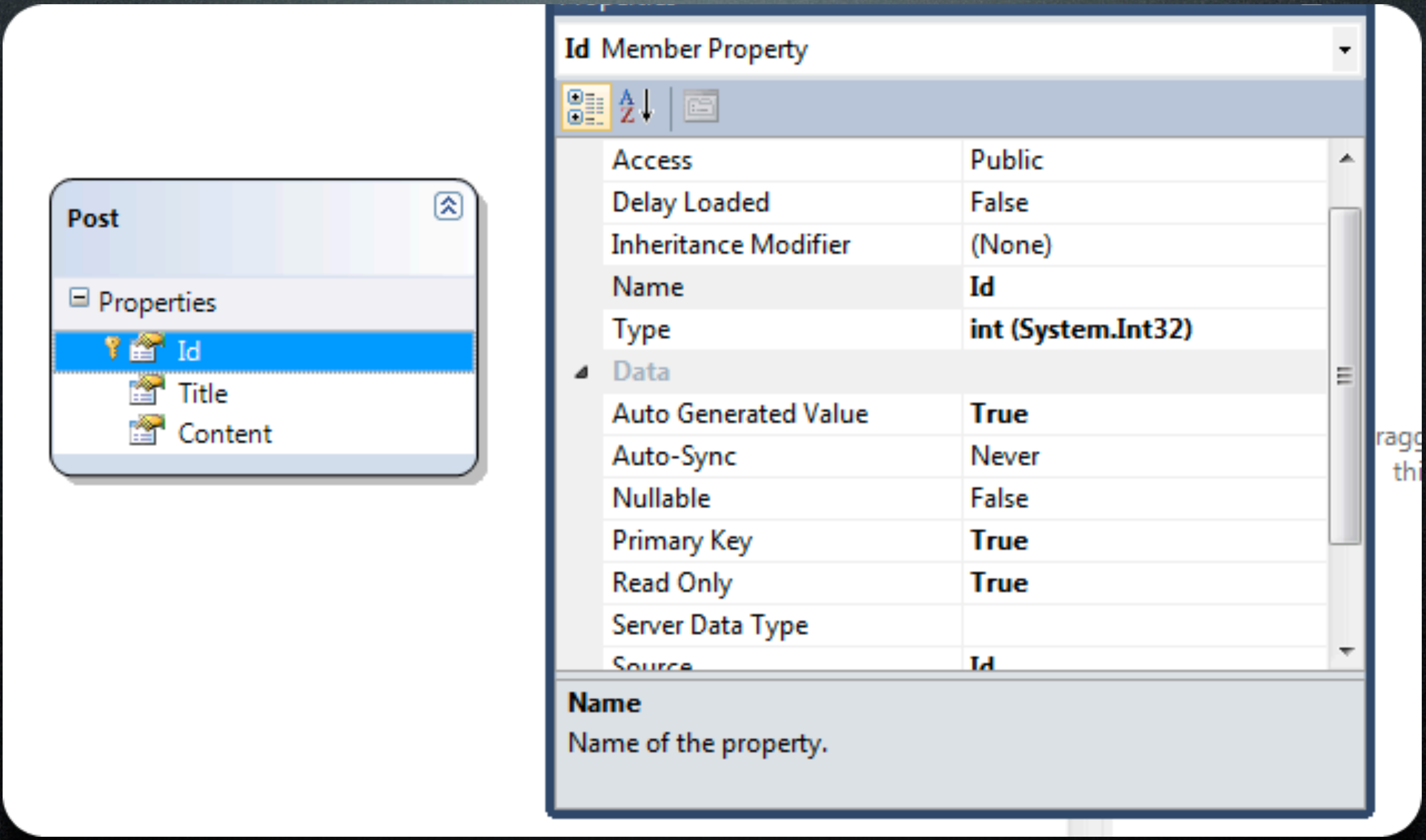
- Url:
  - http://localhost:9165/Post/edit/1
    - this says that we're going to the Post controller, Edit action and passing an id of 1
    - we have no Edit action inside PostController.cs!
    - lets create one

The edit action

- is actually two edit actions
  - 1st action displays the data in an editable way (a form)
  - 2nd action takes the changed data and saves it
- we can copy a lot of the create new post actions
  - 1st we need an edit action
    - create a new method in PostController.cs
      public ViewResult Edit()
    - this is a method that returns a view (html)
    - but wait! we haven't specified an id to load the correct post.

THE MODEL

- we have a pretty simple mental model for this application
- one table - Post, maps to one class- Post
- check out Blog.dbml
- this is a map between classes and tables using “LINQ for SQL”



Post

Properties

Id

Title

Content

Id Member Property

Access	Public
Delay Loaded	False
Inheritance Modifier	(None)
Name	**Id**
Type	**int (System.Int32)**
Data	
Auto Generated Value	**True**
Auto-Sync	Never
Nullable	False
Primary Key	**True**
Read Only	**True**
Server Data Type	
Source	**Id**

Name

Name of the property.

Id is an int

loading the correct Post

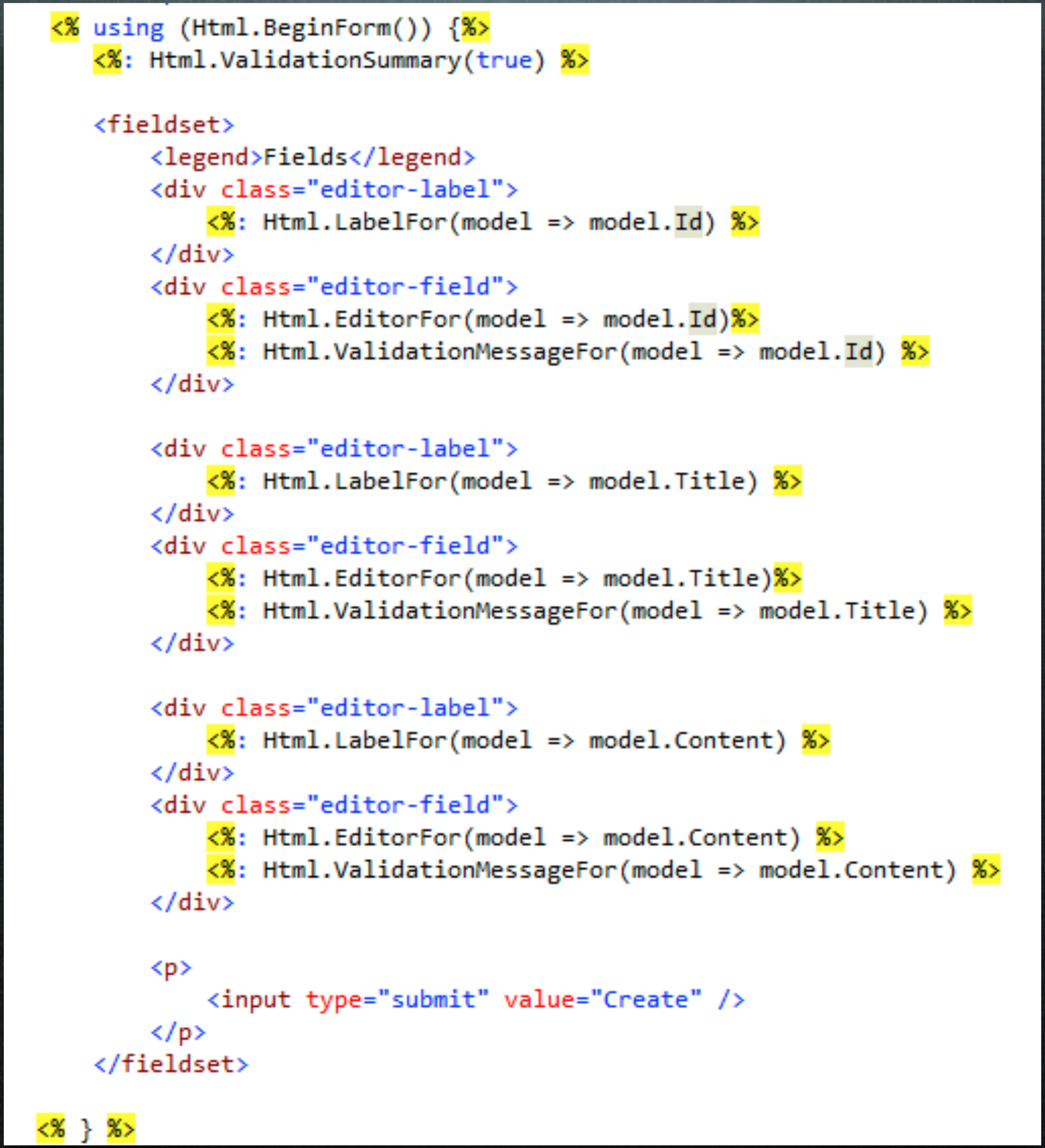
- We already have a field called _blogContext which is our current Database context
- we can load all posts or a specific post
- so take in an int id parameter
- load the correct post
    - var post=_blogContext.Posts.Single(p=>p.Id == id);
- return the View with this post return View (post)

The View

- we now get an error that the edit view doesn’t exist so we need to create it
- right click anywhere in the edit method
  - choose “add view”
  - create a strongly typed view to SummerOfTech_Bootcamp.Models.Post
- save and rerun.. blank page!

Add the Form

- <% is code inline with the html
- Model varable is the model we passed through (an instance of Post)
- we can use helpers Html.EditorFor (Creates an edit box) and Html.LabelFor (adds a text label)
- we start a form with <% using(Html.BeginForm()){%>
- and end it with <%}%>
- try and write an edit box for Title and Content properties for the Post object and put them in a form
- Hint: check out “New.aspx” for the form that is used when we create new posts



```
<% using (Html.BeginForm()) {%>
    <%: Html.ValidationSummary(true) %>

    <fieldset>
        <legend>Fields</legend>
        <div class="editor-label">
            <%: Html.LabelFor(model => model.Id) %>
        </div>
        <div class="editor-field">
            <%: Html.EditorFor(model => model.Id)%>
            <%: Html.ValidationMessageFor(model => model.Id) %>
        </div>

        <div class="editor-label">
            <%: Html.LabelFor(model => model.Title) %>
        </div>
        <div class="editor-field">
            <%: Html.EditorFor(model => model.Title)%>
            <%: Html.ValidationMessageFor(model => model.Title) %>
        </div>

        <div class="editor-label">
            <%: Html.LabelFor(model => model.Content) %>
        </div>
        <div class="editor-field">
            <%: Html.EditorFor(model => model.Content) %>
            <%: Html.ValidationMessageFor(model => model.Content) %>
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>

<% } %>
```

Anyone end up with this?

Try submitting

- nothing saves!
  - we don’t have an action that actually saves the model
  - create another edit method that takes in a Post
  - load the post again and apply the updated Title and Content

Close

- Argh we now have two methods that respond to edit
  - We have to descriminate
    - GET and POST are the difference (load and submit)
    - add an attribute ([Attribute])
    - above each edit method add an [AcceptVerbs] attribute one set to HttpVerbs.Post (the one that saves) and one set to HttpVerbs.Get (the one that loads)

Not quite

- we still get an error!
- Id is read only
- we need a "Wrapper model"
- I've already created one called PostViewModel but we need to make the id writable
  - change public int Id { get; internal set; } to public int Id { get; set; }
  - make the Edit action take a PostViewModel not a Post

UI Niceties

- We can now break things by submitting long titles or no titles
- content should be multi-line not single line
- Id really shouldn’t be shown just put in a hidden field

Easy

- Take a look at the attributes on the fields with PostViewModel
- [HiddenInput], [Required], [StringLength(50)], [DataType(DataType.Text)]
- we also have a nice helper method on Post (ToViewModel) that turns a Post into a PostViewModel
- so in the GET Edit method try returning View (post.ToViewModel()) rather than just the View(post)
  - Hint: you'll have to change the header of the Edit.aspx view page to take a PostViewModel not a Post

Validation

- Still allows us to post bad data (empty or long content/titles)
- we can use ModelState.IsValid to query if the model passed to the method is valid or not...
- try putting the save in an if(ModelState.IsValid) otherwise return the PostViewModel back to the edit view
- we can actually enforece validation at the client side too
    - add the line <%Html.EnableClientValidation(); %> before the form

Bonus

- Now for some challenges:
  - can you get the list of posts to show their content too?
  - can you add a field to the database to save a username against the post and update the create/edit views appropriately?