

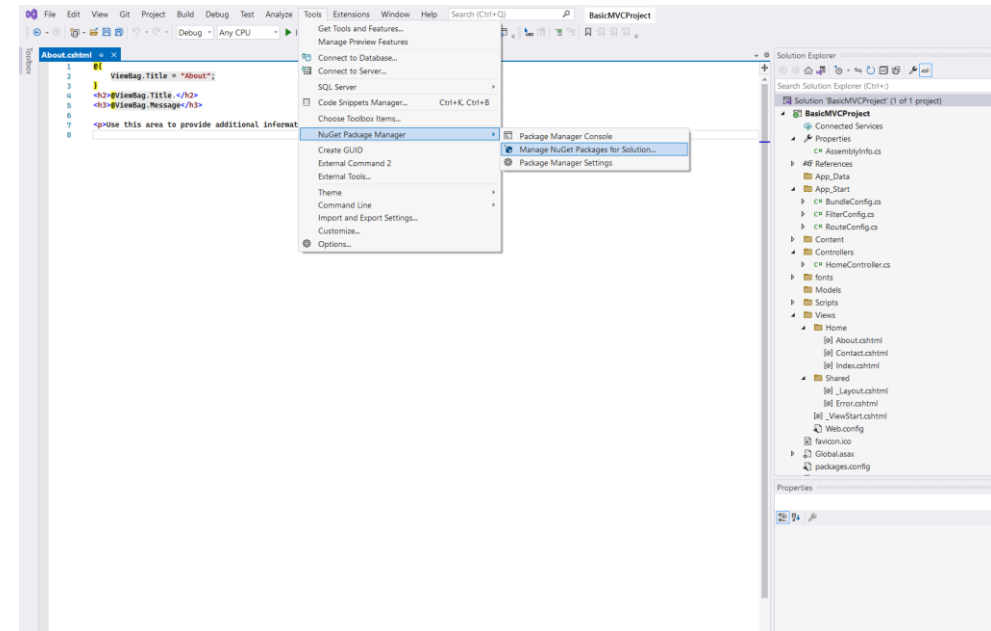
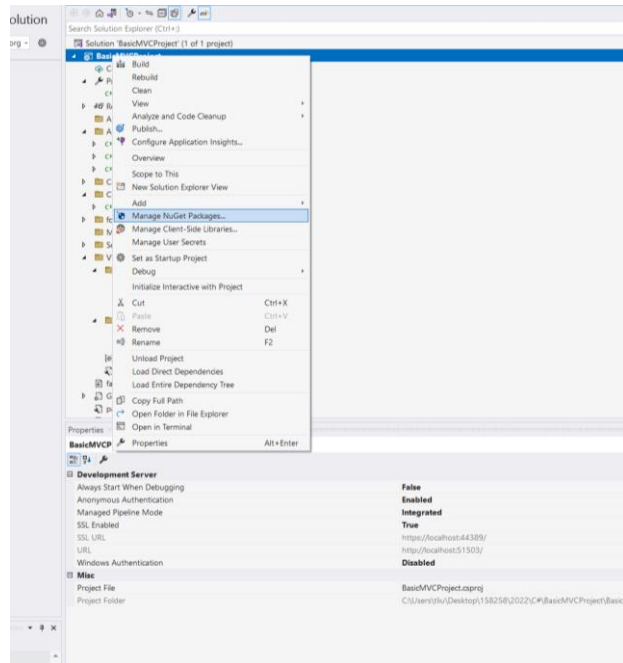
Create an ASP.net MVC project  
using Entity Framework

# Creating Views, Controllers, and a Database from Model Classes

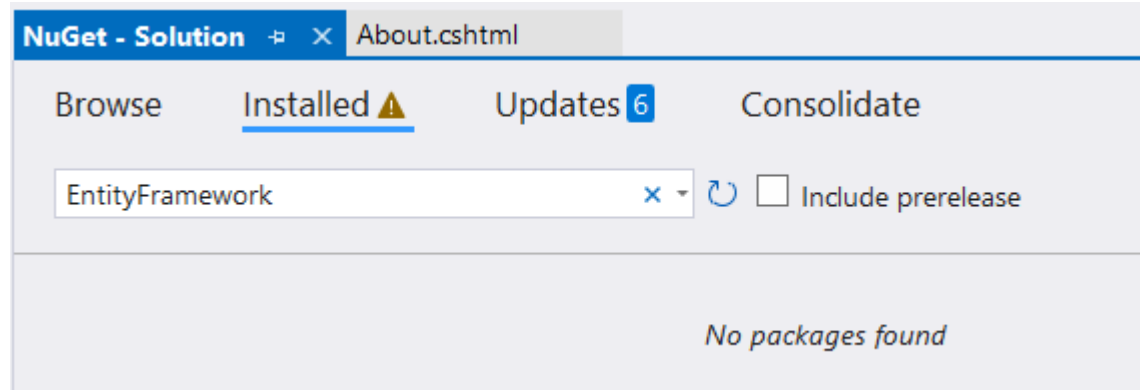
- Create a database using Entity Framework Code First.
  - create a database from your code
  - add new views and controllers to manage and display data

# Entity Framework NuGet package

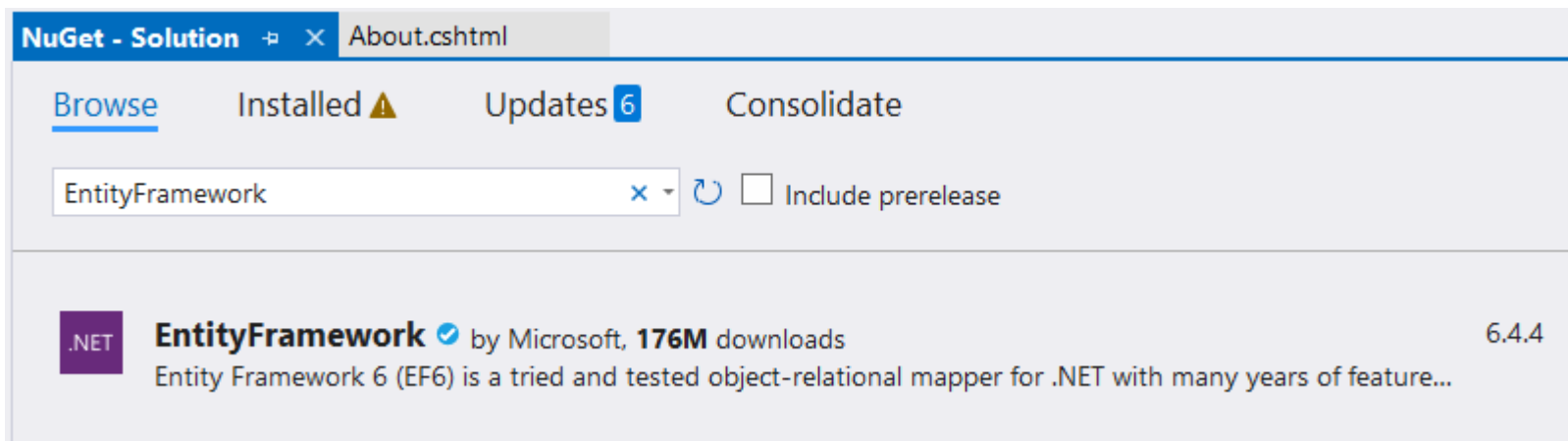
- If your Visual Studio doesn't have entity framework already installed then you need to search for it and install it;
  - by right clicking your project file and clicking Manage NuGet Packages.
  - Or Tools-> NuGet Package Manager -> Manage NuGet Packages for solution



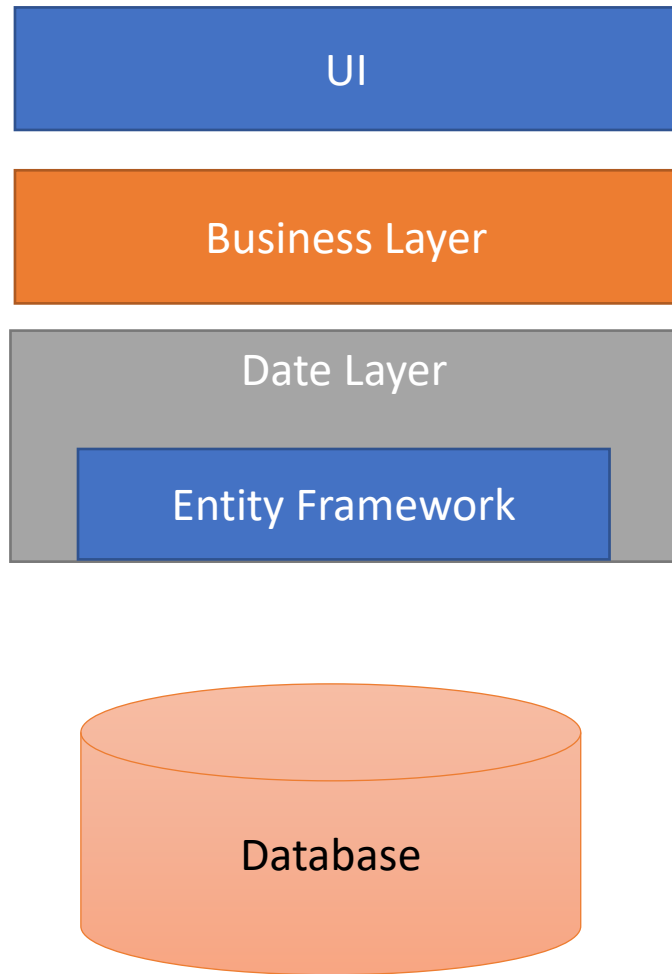
# Entity Framework Nuget package



- It will show you the installed ones and the one you can browse and install.
- Make sure you have the latest one



# Entity Framework



- Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.
- Entity Framework Code First allows the creation of a database from classes.
- EF in a code first approach enables developers to write classes and relationships between them in code to model and build a database without writing any SQL.

# Entity Framework

## **Why?**

- 1. Cross-platform**
- 2. Provide Excellent Prototypes.**
- 3. Ability to Shorten Codes.**
- 4. Auto-Migrations**

## **Issues:**

- 1. Can Complicate Things.**
- 2. Dictates the Model Shape.**

# DbContext

DbContext is the primary class that is responsible for interacting with data as object. DbContext contains entity set (DbSet<TEntity>) for all the entities which is mapped to DB tables. DbContext is responsible for the following activities:

- Write and execute queries
- Materialize query results as entity objects
- Track changes that are made to those objects
- Persist object changes back on the database
- Bind objects in memory to UI controls

# Product

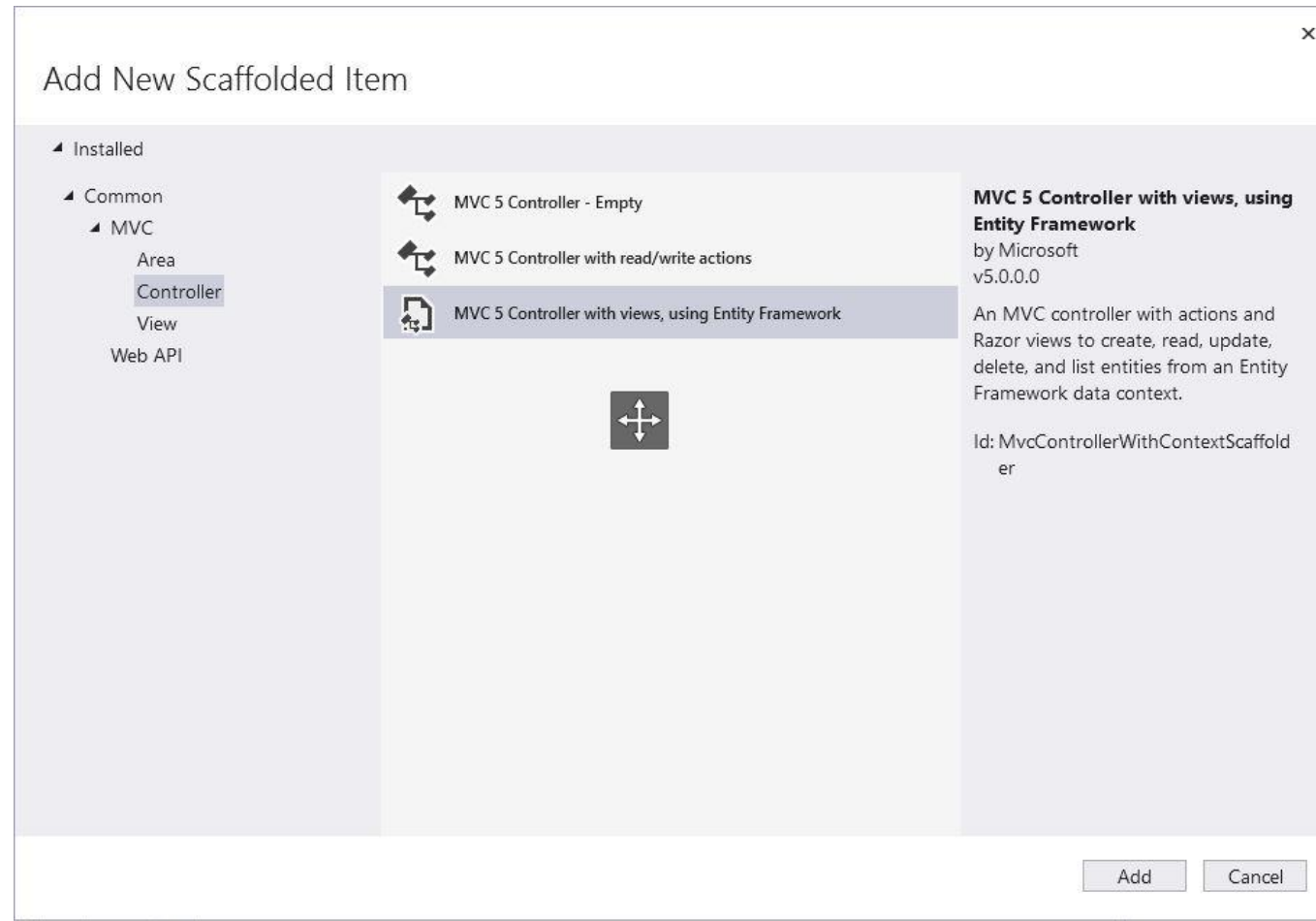
- `using System.ComponentModel.DataAnnotations;`
- `namespace BasicMVCProject.Models`
- `{`
- `public class Product`
- `{`
- `[Key]`
- `public int ID { get; set; }`
- `public string Name { get; set; }`
- `public decimal Price { get; set; }`
- `public string Description { get; set; }`
- `public int? CategoryID { get; set; }`
- `public virtual Category Category { get; set; }`
- `}`
- `}`



# Category

- `using System.Collections;`
- `using System.Collections.Generic;`
- `using System.ComponentModel.DataAnnotations;`
  
- `namespace BasicMVCProject.Models`
- `{`
- `public class Category`
- `{`
- `[Key]`
- `public int ID { get; set; }`
- `public string Name { get; set; }`
- `public virtual ICollection<Product> Products { get; set; }`
- `}`
- `}`

# Adding Controller



# Adding Controller


Add Controller ✕

Model class:

Data context class:  +

☐ Use async controller actions

Views:

☒ Generate views 

☒ Reference script libraries

☒ Use a layout page:

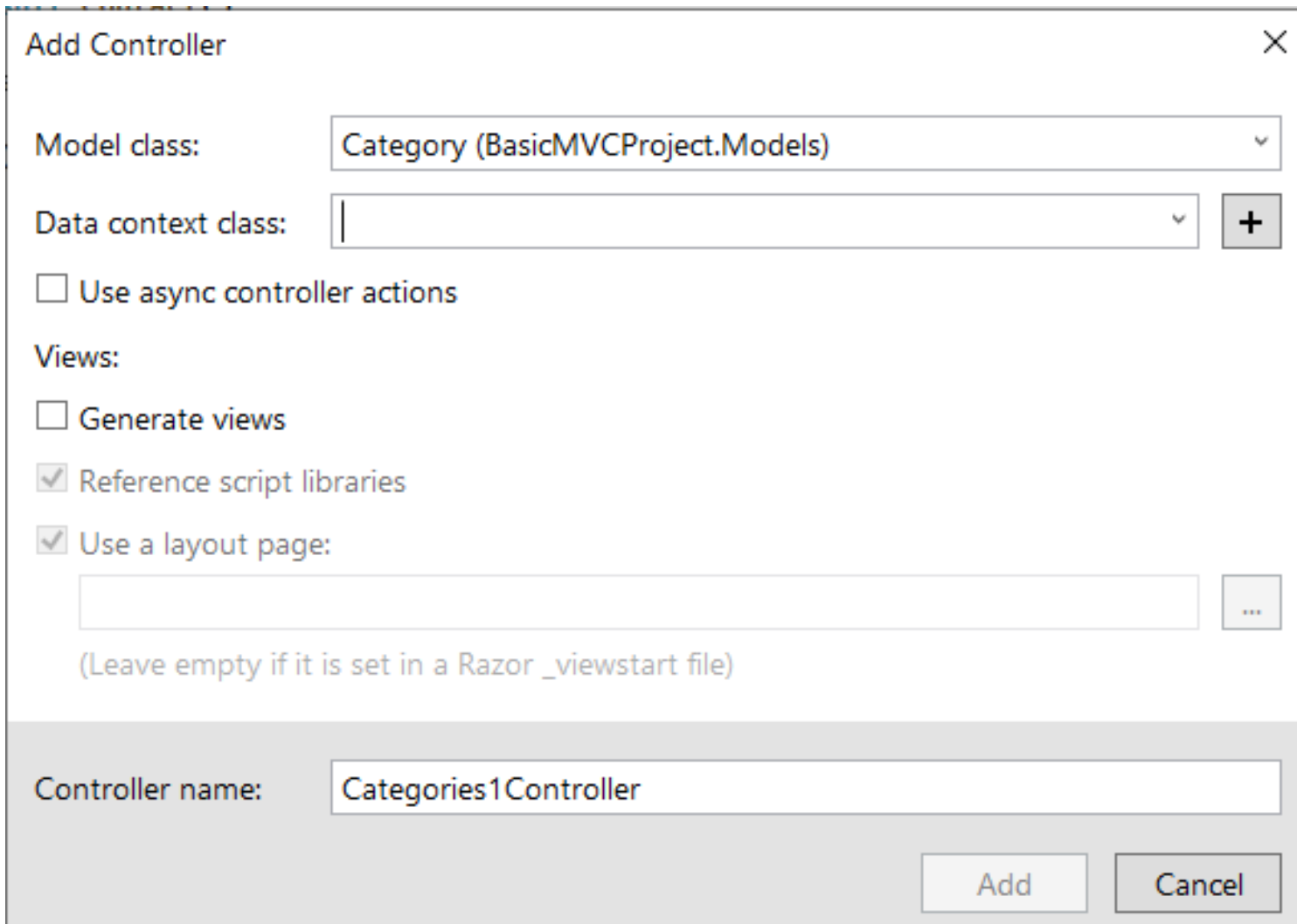
...

(Leave empty if it is set in a Razor \_viewstart file)

Controller name:

Add Cancel

# Adding Controller



The image shows a 'Add Controller' dialog box with a close button (X) in the top right corner. It contains several input fields and checkboxes. The 'Model class' dropdown is set to 'Category (BasicMVCProject.Models)'. The 'Data context class' dropdown is empty, with a '+' button to its right. There are three checkboxes: 'Use async controller actions' (unchecked), 'Generate views' (unchecked), and 'Reference script libraries' (checked). Below these is another checked checkbox 'Use a layout page:' followed by an empty text box and a '...' button. A note below the text box says '(Leave empty if it is set in a Razor \_viewstart file)'. At the bottom, the 'Controller name' text box contains 'Categories1Controller'. The bottom right has 'Add' and 'Cancel' buttons.

Add Controller

Model class: Category (BasicMVCProject.Models)

Data context class: +

☐ Use async controller actions

Views:

☐ Generate views

☒ Reference script libraries

☒ Use a layout page:

(Leave empty if it is set in a Razor \_viewstart file)

Controller name: Categories1Controller

Add Cancel

# Adding Controller

Add Controller

Model class:

Category (BasicMVCProject.Models)

Data context class:

BasicMVCProjectContext (BasicMVCProject.Data)

+

☐ Use async controller actions

Views:

☐ Generate views

☒ Reference script libraries

☒ Use a layout page:

...

(Leave empty if it is set in a Razor \_viewstart file)

Controller name:

Categories1Controller

Add

Cancel

# CategoriesController.cs

```
public class CategoriesController : Controller
{
    private BasicMVCProjectContext db = new BasicMVCProjectContext();

    // GET: Categories
    public ActionResult Index()
    {
        return View(db.Categories.ToList());
    }

    // GET: Categories/Details/5
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Category category = db.Categories.Find(id);
        if (category == null)
        {
            return HttpNotFound();
        }
        return View(category);
    }

    // GET: Categories/Create
    public ActionResult Create()
    {
        return View();
    }
}
```

# CategoriesController.cs

```
public class CategoriesController : Controller
{
    private BasicMVCProjectContext db = new BasicMVCProjectContext();

    // POST: Categories/Create
    // To protect from overposting attacks, enable the specific properties you want to bind to, for
    // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create([Bind(Include = "ID,Name")] Category category)
    {
        if (ModelState.IsValid)
        {
            db.Categories.Add(category);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(category);
    }

    // GET: Categories/Edit/5
    public ActionResult Edit(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Category category = db.Categories.Find(id);
        if (category == null)
        {
            return HttpNotFound();
        }
        return View(category);
    }
}
```

# CategoriesController.cs

```
public class CategoriesController : Controller
{
    private BasicMVCProjectContext db = new BasicMVCProjectContext();

    // POST: Categories/Edit/5
    // To protect from overposting attacks, enable the specific properties you want to bind to, for
    // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit([Bind(Include = "ID,Name")] Category category)
    {
        if (ModelState.IsValid)
        {
            db.Entry(category).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(category);
    }

    // GET: Categories/Delete/5
    public ActionResult Delete(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Category category = db.Categories.Find(id);
        if (category == null)
        {
            return HttpNotFound();
        }
        return View(category);
    }

    // POST: Categories/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public ActionResult DeleteConfirmed(int id)
    {
        Category category = db.Categories.Find(id);
        db.Categories.Remove(category);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            db.Dispose();
        }
        base.Dispose(disposing);
    }
}
```



# Database Context: DbContext

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace BasicMVCProject.Data
{
    5 references
    public class BasicMVCProjectContext : DbContext
    {
        // You can add custom code to this file. Changes will not be overwritten.
        //
        // If you want Entity Framework to drop and regenerate your database
        // automatically whenever you change your model schema, please use data migrations.
        // For more information refer to the documentation:
        // http://msdn.microsoft.com/en-us/data/jj591621.aspx

        2 references
        public BasicMVCProjectContext() : base("name=BasicMVCProjectContext")
        {
        }

        7 references
        public System.Data.Entity.DbSet<BasicMVCProject.Models.Product> Products { get; set; }

        11 references
        public System.Data.Entity.DbSet<BasicMVCProject.Models.Category> Categories { get; set; }
    }
}
```

Search Solution Explorer (Ctrl+;)

- Content
- ▾ Controllers
  - C# CategoriesController.cs
  - C# HomeController.cs
  - C# ProductsController.cs
- ▾ Data
  - C# BasicMVCProjectContext.cs
- fonts
- ▾ Models
  - C# Category.cs
  - C# Product.cs
- Scripts
- ▾ Views
  - ▾ Categories
    - [@] Create.cshtml
    - [@] Delete.cshtml
    - [@] Details.cshtml
    - [@] Edit.cshtml
    - [@] Index.cshtml
  - ▾ Home
    - [@] About.cshtml

# Database Context: DbContext

```
using System.Data.Entity;

namespace BasicMVCProject.Data
{
    public class BasicMVCProjectContext : DbContext

        public BasicMVCProjectContext() : base("name=BasicMVCProjectContext")
        {
        }

        public System.Data.Entity.DbSet<BasicMVCProject.Models.Product> Products { get; set; }

        public System.Data.Entity.DbSet<BasicMVCProject.Models.Category> Categories { get; set; }
    }
}
```

# Connection String

```
<connectionStrings>  
    <add name="BasicMVCProjectContext" connectionString="Data  
Source=(localdb)\MSSQLLocalDB; Initial Catalog=BasicMVCProjectContext-  
20220910231737; Integrated Security=True; MultipleActiveResultSets=True;  
AttachDbFilename=|DataDirectory|BasicMVCProjectContext-20220910231737.mdf"  
        providerName="System.Data.SqlClient" />  
</connectionStrings>
```

# Adding Views

```
Layout.cshtml Web.config BasicMVCPr...tContext.cs CategoriesController.cs ProductsController.cs Category.cs Product.cs
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>@ViewBag.Title - My ASP.NET Application</title>
7 @Styles.Render("~/Content/css")
8 @Scripts.Render("~/bundles/modernizr")
9 </head>
10 <body>
11 <div class="navbar navbar-inverse navbar-fixed-top">
12 <div class="container">
13 <div class="navbar-header">
14 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse" title="more options">
15 <span class="icon-bar"></span>
16 <span class="icon-bar"></span>
17 <span class="icon-bar"></span>
18 </button>
19 @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
20 </div>
21 <div class="navbar-collapse collapse">
22 <ul class="nav navbar-nav">
23 <li>@Html.ActionLink("Home", "Index", "Home")</li>
24 <li>@Html.ActionLink("About", "About", "Home")</li>
25 <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
26 <li>@Html.ActionLink("Category", "Index", "Categories")</li>
27 <li>@Html.ActionLink("Product", "Index", "Products")</li>
28 </ul>
29 </div>
30 </div>
31 </div>
32 <div class="container body-content">
33 @RenderBody()
34 <hr />
35 <footer>
36 <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
37 </footer>
38 </div>
39
40 @Scripts.Render("~/bundles/jquery")
```

## Index

[Create New](#)

Name
Accessories

[Edit](#) | [Details](#) | [Delete](#)

# Lambda Expressions

- create an anonymous function
- Expression Lambda: Consists of the input and the expression.
- Syntax: (input-parameters) => expression
- Statement Lambda: Consists of the input and a set of statements to be executed.
- Syntax: (input-parameters) => { <sequence-of-statements> }

# Lambda Expressions Example

$$f(x) = x * x$$

or

$$\text{square}(x) = x * x$$

C# Lambda Expressions:

$$x \Rightarrow x * x$$

# Binding view to the Model: ViewData

- `@model`: is a keyword reserved by Razor. It states that the current Model type associated with this View.
- You can then access the instance of this Model type by using the Model keyword. (Capitalisation in C# matters)
- Model with a capital M means a call to the instance of Model and model represents the declaration (similar to using)
- Model is strongly typed and you can access the properties of your model class by simply stating the property name.
- If you want a list of all your members, you use `IEnumerable` of your model.



# Views\Categories\Index.cshtml

```
1  @model IEnumerable<BasicMVCProject.Models.Category>
2
3  @{
4      ViewBag.Title = "Index";
5  }
6
7  <h2>Index</h2>
8
9  <p>
10     @Html.ActionLink("Create New", "Create")
11 </p>
12 <table class="table">
13 <tr>
14 <th>
15     @Html.DisplayNameFor(model => model.Name)
16 </th>
17 <th></th>
18 </tr>
19
20 @foreach (var item in Model) {
21 <tr>
22 <td>
23     @Html.DisplayFor(modelItem => item.Name)
24 </td>
25 <td>
26     @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
27     @Html.ActionLink("Details", "Details", new { id=item.ID }) |
28     @Html.ActionLink("Delete", "Delete", new { id=item.ID })
29 </td>
30 </tr>
31 }
32
33 </table>
34
```

**Solution Explorer**

Search Solution Explorer (Ctrl+;)

- C# ProductsController.cs
- Data
- C# BasicMVCProjectContext.cs
- fonts
- Models
  - C# Category.cs
  - C# Product.cs
- Scripts
- Views
  - Categories
    - [@] Create.cshtml
    - [@] Delete.cshtml
    - [@] Details.cshtml
    - [@] Edit.cshtml
    - [@] Index.cshtml
  - Home
    - [@] About.cshtml
    - [@] Contact.cshtml
    - [@] Index.cshtml
  - Products
    - [@] Create.cshtml

**Properties**

# Views\Categories\Details.cshtml

The image shows a Visual Studio editor window with the file `Details.cshtml` open. The code is as follows:

```
1 @model BasicMVCProject.Models.Category
2
3 @{
4     ViewBag.Title = "Details";
5 }
6
7 <h2>Details</h2>
8
9 <div>
10     <h4>Category</h4>
11     <hr />
12     <dl class="dl-horizontal">
13         <dt>
14             @Html.DisplayNameFor(model => model.Name)
15         </dt>
16
17         <dd>
18             @Html.DisplayFor(model => model.Name)
19         </dd>
20     </dl>
21 </div>
22
23 <p>
24     @Html.ActionLink("Edit", "Edit", new { id = Model.ID }) |
25     @Html.ActionLink("Back to List", "Index")
26 </p>
27
```

The Solution Explorer on the right shows the project structure:

- C# ProductsController.cs
- Data
  - BasicMVCProjectContext.cs
- fonts
- Models
  - Category.cs
  - Product.cs
- Scripts
- Views
  - Categories
    - Create.cshtml
    - Delete.cshtml
    - Details.cshtml
    - Edit.cshtml
    - Index.cshtml
  - Home
    - About.cshtml
    - Contact.cshtml
    - Index.cshtml
  - Products
    - Create.cshtml

# Views\Categories\Create.cshtml

The image shows a Visual Studio IDE with the 'Create.cshtml' file open in the editor. The code is a Razor view for creating a new category. It includes a title, a validation summary, a form with a label and an editor for the 'Name' property, and a submit button. The view also includes a link to the 'Index' page and a section for scripts.

```
1 @model BasicMVCProject.Models.Category
2
3 @{
4     ViewBag.Title = "Create";
5 }
6
7 <h2>Create</h2>
8
9
10 @using (Html.BeginForm())
11 {
12     @Html.AntiForgeryToken()
13
14     <div class="form-horizontal">
15         <h4>Category</h4>
16         <hr />
17         @Html.ValidationSummary(true, "", new { @class = "text-danger" })
18         <div class="form-group">
19             @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
20             <div class="col-md-10">
21                 @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
22                 @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
23             </div>
24         </div>
25
26         <div class="form-group">
27             <div class="col-md-offset-2 col-md-10">
28                 <input type="submit" value="Create" class="btn btn-default" />
29             </div>
30         </div>
31     </div>
32 }
33
34 <div>
35     @Html.ActionLink("Back to List", "Index")
36 </div>
37
38 @section Scripts {
39     @Scripts.Render("~/bundles/jqueryval")
40 }
41
```

The Solution Explorer on the right shows the project structure. The 'Views' folder is expanded, showing the 'Categories' subfolder. The 'Create.cshtml' file is selected and highlighted.

Properties

# Views\Categories\Edit.cshtml

The image shows a Visual Studio IDE with the `Edit.cshtml` file open in the editor. The code is a Razor view for editing a category. It includes a model declaration, a title setting, a heading, a form with validation, and a back link.

```
1 @model BasicMVCProject.Models.Category
2
3 @{
4     ViewBag.Title = "Edit";
5 }
6
7 <h2>Edit</h2>
8
9
10 @using (Html.BeginForm())
11 {
12     @Html.AntiForgeryToken()
13
14     <div class="form-horizontal">
15         <h4>Category</h4>
16         <hr />
17         @Html.ValidationSummary(true, "", new { @class = "text-danger" })
18         @Html.HiddenFor(model => model.ID)
19
20         <div class="form-group">
21             @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
22             <div class="col-md-10">
23                 @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
24                 @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
25             </div>
26         </div>
27
28         <div class="form-group">
29             <div class="col-md-offset-2 col-md-10">
30                 <input type="submit" value="Save" class="btn btn-default" />
31             </div>
32         </div>
33     </div>
34 }
35
36 <div>
37     @Html.ActionLink("Back to List", "Index")
38 </div>
39
40 @section Scripts {
41     @Scripts.Render("~/bundles/jqueryval")
42 }
43
```

The Solution Explorer on the right shows the project structure:

- C# ProductsController.cs
- Data
  - C# BasicMVCProjectContext.cs
- fonts
- Models
  - C# Category.cs
  - C# Product.cs
- Scripts
- Views
  - Categories
    - [@] Create.cshtml
    - [@] Delete.cshtml
    - [@] Details.cshtml
    - [@] Edit.cshtml
    - [@] Index.cshtml
  - Home
    - [@] About.cshtml
    - [@] Contact.cshtml
    - [@] Index.cshtml
  - Products
    - [@] Create.cshtml

The Properties window at the bottom is empty.

# Views\Categories\Delete.cshtml

The image shows a Visual Studio editor window with the file `Delete.cshtml` open. The code is a Razor view for deleting a category. It includes a model of type `BasicMVCProject.Models.Category`, sets the view bag title to "Delete", and displays the category name. It also contains a confirmation message and a form with a submit button and a "Back to List" link.

```
1 @model BasicMVCProject.Models.Category
2
3 @{
4     ViewBag.Title = "Delete";
5 }
6
7 <h2>Delete</h2>
8
9 <h3>Are you sure you want to delete this?</h3>
10 <div>
11     <h4>Category</h4>
12     <hr />
13     <dl class="dl-horizontal">
14         <dt>
15             @Html.DisplayNameFor(model => model.Name)
16         </dt>
17         <dd>
18             @Html.DisplayFor(model => model.Name)
19         </dd>
20     </dl>
21
22     @using (Html.BeginForm()) {
23         @Html.AntiForgeryToken()
24
25         <div class="form-actions no-color">
26             <input type="submit" value="Delete" class="btn btn-default" /> |
27             @Html.ActionLink("Back to List", "Index")
28         </div>
29     }
30 </div>
```

The Solution Explorer on the right shows the project structure. The `Views` folder is expanded, showing the `Categories` subfolder. The `Delete.cshtml` file is selected and highlighted.

**Solution Explorer:**

- C# ProductsController.cs
- Data
- C# BasicMVCProjectContext.cs
- fonts
- Models
- C# Category.cs
- C# Product.cs
- Scripts
- Views
  - Categories
    - [@] Create.cshtml
    - [@] Delete.cshtml
    - [@] Details.cshtml
    - [@] Edit.cshtml
    - [@] Index.cshtml
  - Home
    - [@] About.cshtml
    - [@] Contact.cshtml
    - [@] Index.cshtml
  - Products
    - [@] Create.cshtml

# Categories table in the database

The screenshot displays the SQL Server Enterprise Designer interface. On the left, the SQL Server Object Explorer shows the database structure, with the **dbo.Categories** table selected. The main pane shows the table design for **dbo.Categories** in Design view. The table has two columns: **ID** (int, primary key, not null) and **Name** (nvarchar(MAX), nullable). The right pane shows the table's properties, including a primary key constraint **PK\_dbo.Categories**. The bottom pane shows the T-SQL script for creating the table.

Name	Data Type	Allow Nulls	Default
ID	int	<input type="checkbox"/>	
Name	nvarchar(MAX)	<input checked="" type="checkbox"/>	

**Keys (1)**  
PK\_dbo.Categories (Primary Key, Clustered)

**Check Constraints (0)**  
**Indexes (0)**  
**Foreign Keys (0)**  
**Triggers (0)**

```
1 CREATE TABLE [dbo].[Categories] (  
2     [ID] INT IDENTITY (1, 1) NOT NULL,  
3     [Name] NVARCHAR (MAX) NULL,  
4     CONSTRAINT [PK_dbo.Categories] PRIMARY KEY CLUSTERED ([ID] ASC)  
5 )
```

# Viewing the data in the Categories table in the database

