Asp.Net Core MVC

Jim Fawcett
CSE686 – Internet Programming
Spring 2019

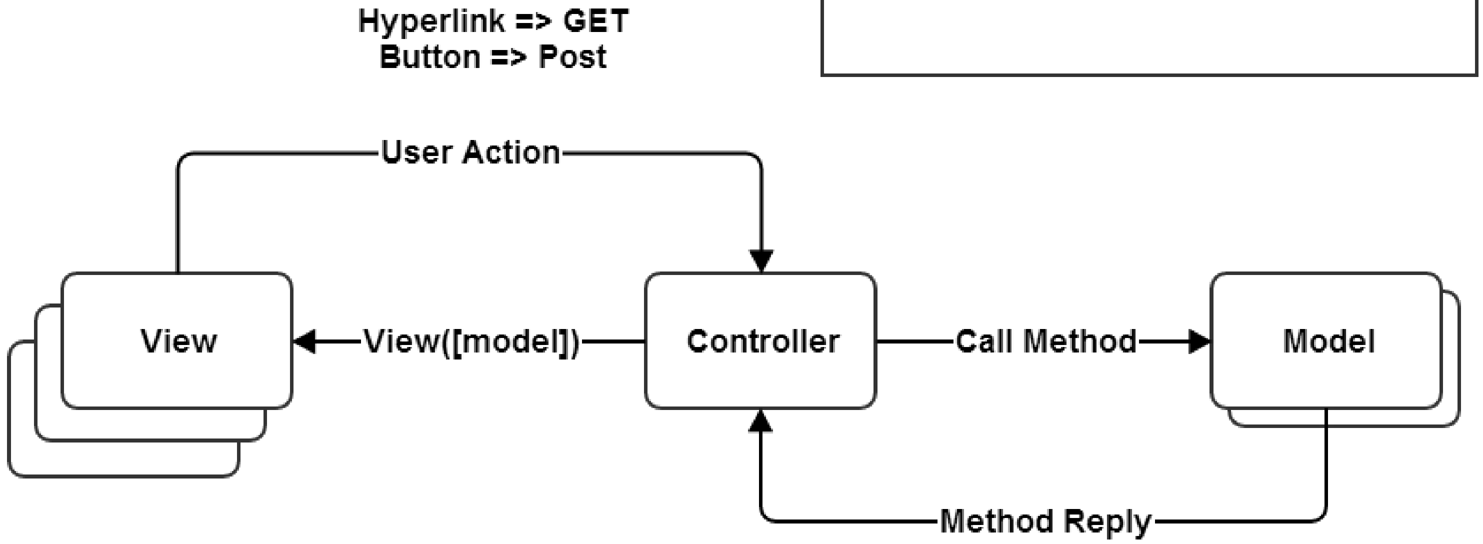
What is Asp.Net Core MVC?

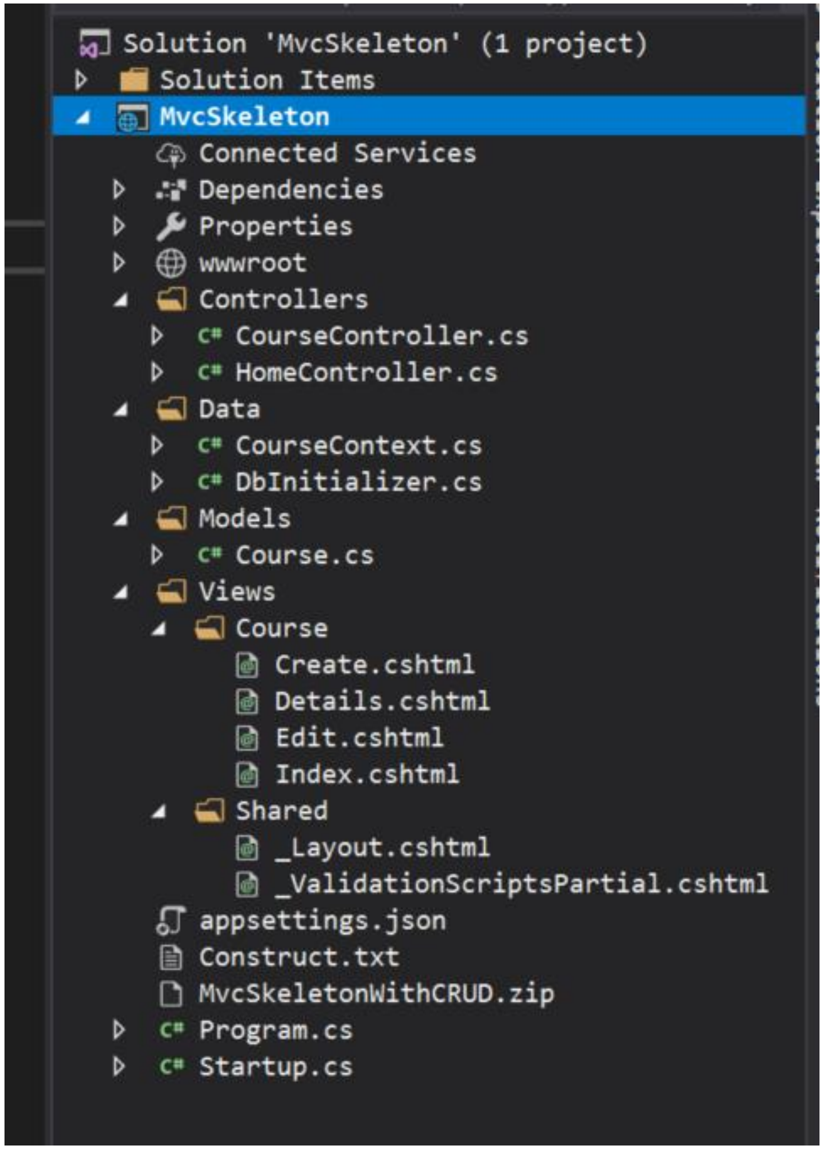
- Framework for building web applications
- Based on Model-View-Controller pattern
  - Model manages the application data and enforces constraints on that model.
    - Often accessed through persistent objects
  - Views are mostly passive presentations of application state.
    - Views generate requests sent to a controller based on client actions.
  - Controllers translate requests into actions on the data model and generate subsequent views.

MVC Structure

GET Request often returns selected item id as query string

POST returns model values in FormCollection





```
Solution 'MvcSkeleton' (1 project)
  Solution Items
  MvcSkeleton
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      CourseController.cs
      HomeController.cs
    Data
      CourseContext.cs
      DbInitializer.cs
    Models
      Course.cs
    Views
      Course
        Create.cshtml
        Details.cshtml
        Edit.cshtml
        Index.cshtml
      Shared
        _Layout.cshtml
        _ValidationScriptsPartial.cshtml
    appsettings.json
    Construct.txt
    MvcSkeletonWithCRUD.zip
    Program.cs
    Startup.cs
```

Mvc Structure

- Controllers
  - Connect Views to Data
- Models
  - Provide structured data, usually persisted to a db
  - Accessed through C# class instances
- Views
  - Combine markup and C# code to display and accept data.

MVC Life Cycle

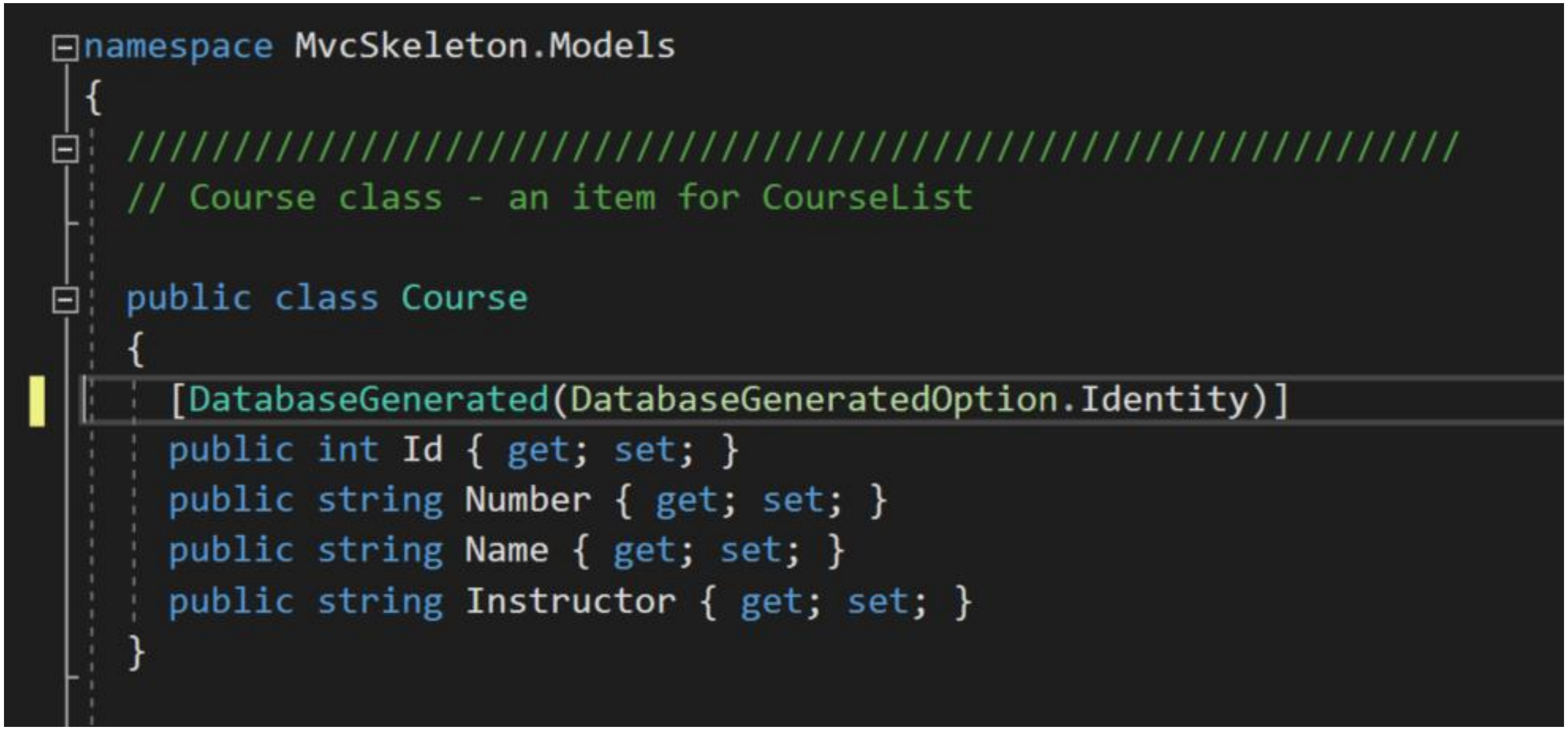
- Clients request a named action on a specified controller, e.g.:
  - http://localhost/aController/anAction

- The request is routed to aController’s anAction method.
  - That method decides how to handle the request, perhaps by accessing a model’s state and returning some information in a view.
  - User actions in the view, e.g., data entered, button presses, result in get (ActionLink) or post (Button) requests to a specific controller action.
  - That process may repeat for many cycles.

What is a Model?

- A model is a file of C# code and often an associated data store, e.g., an SQL database or XML file.
  - The file of C# code manages all access to the application’s data through objects.
  - Linq to SQL and Linq to XML can be used to create queries into these data stores
    - This can be direct
    - More often it is done through objects that wrap db tables or XML files and have one public property for each attribute column of the table.

MvcSkeleton with CRUD Model



```
namespace MvcSkeleton.Models
{
    ////////////////////////////////////////////////////////////////
    // Course class - an item for CourseList

    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        public string Number { get; set; }
        public string Name { get; set; }
        public string Instructor { get; set; }
    }
```

Adding a Model

- Right-click on Model folder and select Add Class.
  - Populate the model class with public properties that represent data to be managed.
  - Usually the model is persisted to an XML file or SQL database using LINQ or the Entity Data Framework.

What is a View?

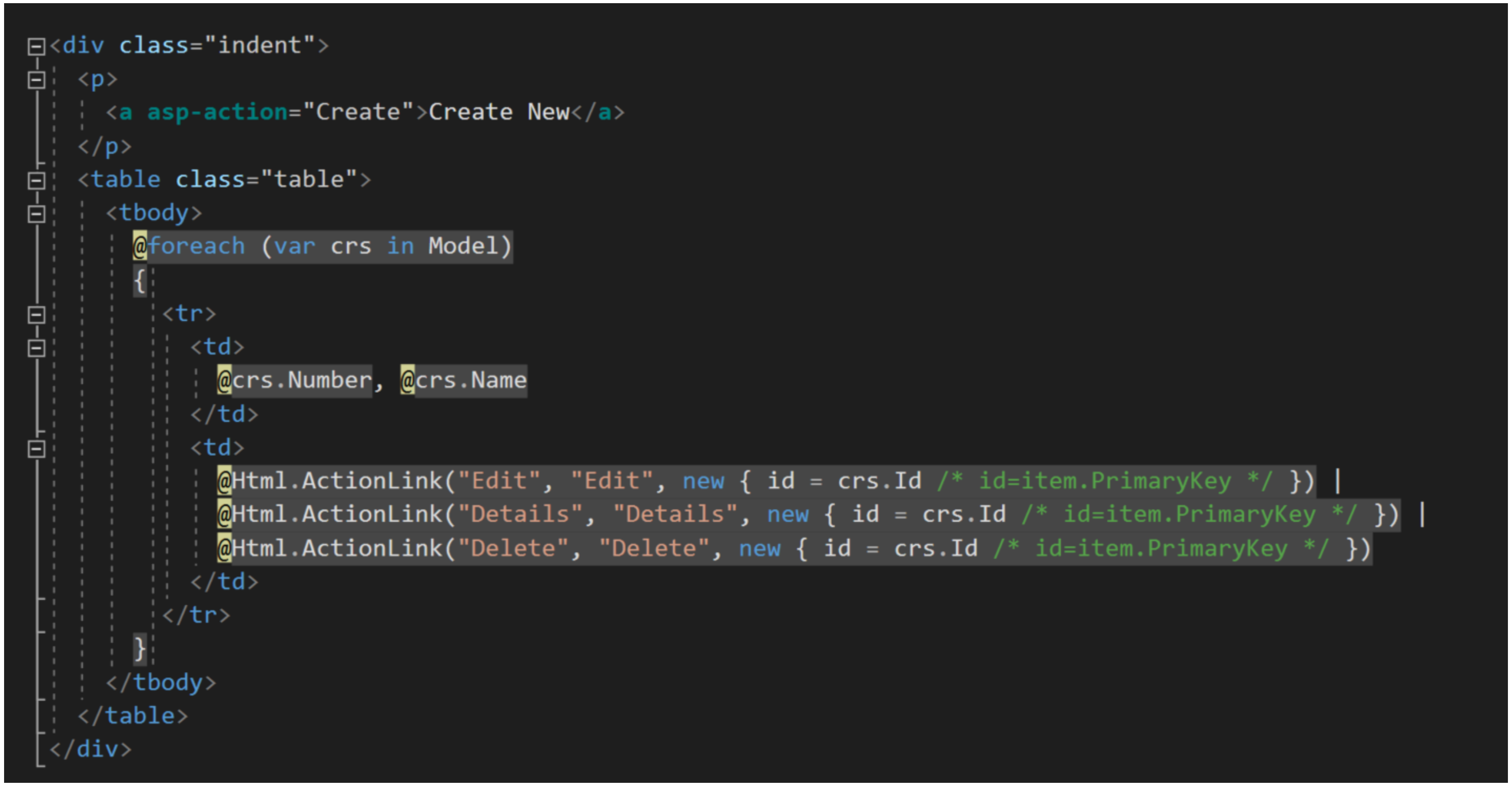
- Views are cshtml files with only HTML and inline C# code, e.g., <td>@crs.Number, @crs.Name
  - Code is used just to support presentation and does no application processing.
  - The HTML is augmented by HTML Helpers, provided by Asp.Net Core MVC that provide shortcuts for commonly used HTML constructs, e.g.:

    ```
    @Html.ActionLink(“Edit”, “Edit”, new { id = crs.Id })
    ```

  - Asp.Net MVC also provides tag helpers that translate into pure markup, e.g.:

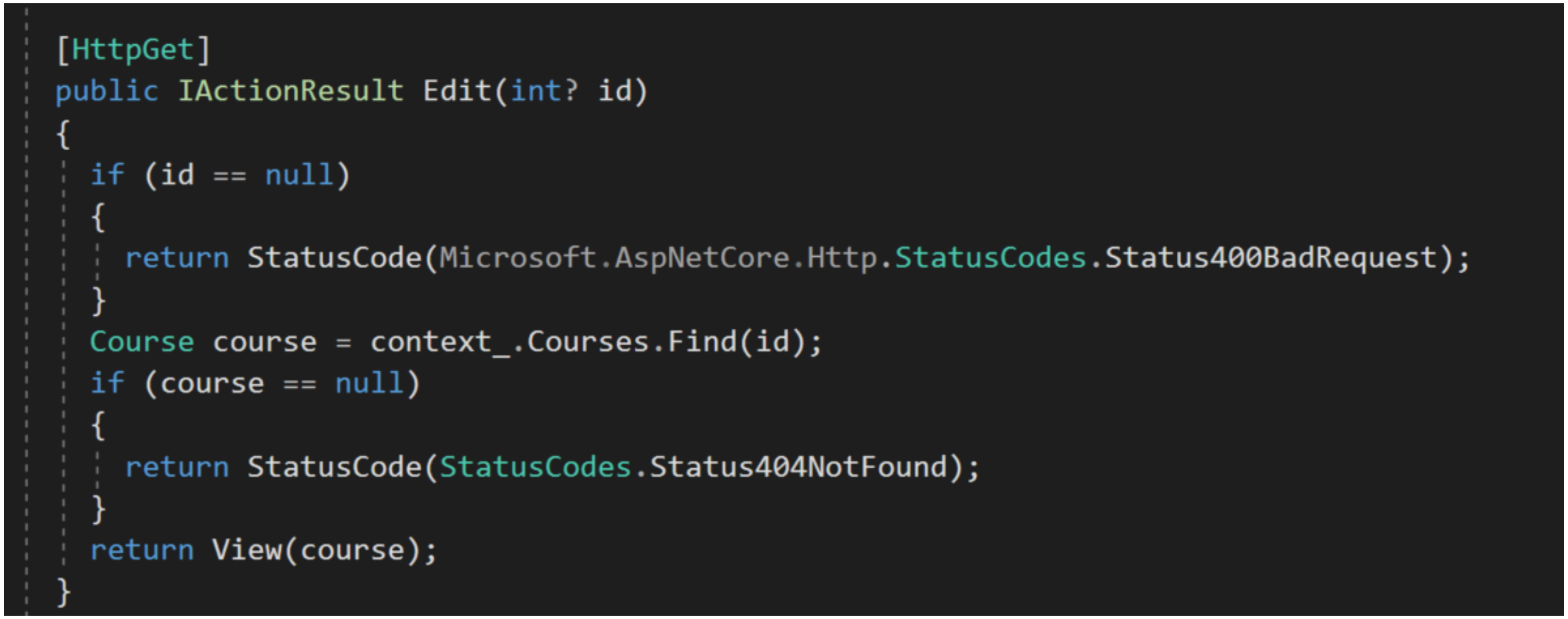
    ```
    <input asp-for=“Name” />
    ```

Create View



```
<div class="indent">
    <p>
        <a asp-action="Create">Create New</a>
    </p>
    <table class="table">
        <tbody>
            @foreach (var crs in Model)
            {
                <tr>
                    <td>
                        @crs.Number, @crs.Name
                    </td>
                    <td>
                        @Html.ActionLink("Edit", "Edit", new { id = crs.Id /* id=item.PrimaryKey */ }) |
                        @Html.ActionLink("Details", "Details", new { id = crs.Id /* id=item.PrimaryKey */ }) |
                        @Html.ActionLink("Delete", "Delete", new { id = crs.Id /* id=item.PrimaryKey */ })
                    </td>
                </tr>
            }
        </tbody>
    </table>
</div>
```

Views are results of Controller actions (methods)



```
[HttpGet]
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return StatusCode(Microsoft.AspNetCore.Http.StatusCodes.Status400BadRequest);
    }
    Course course = context_.Courses.Find(id);
    if (course == null)
    {
        return StatusCode(StatusCodes.Status404NotFound);
    }
    return View(course);
}
```

Html Helpers

- ActionLink:
  links to an action method

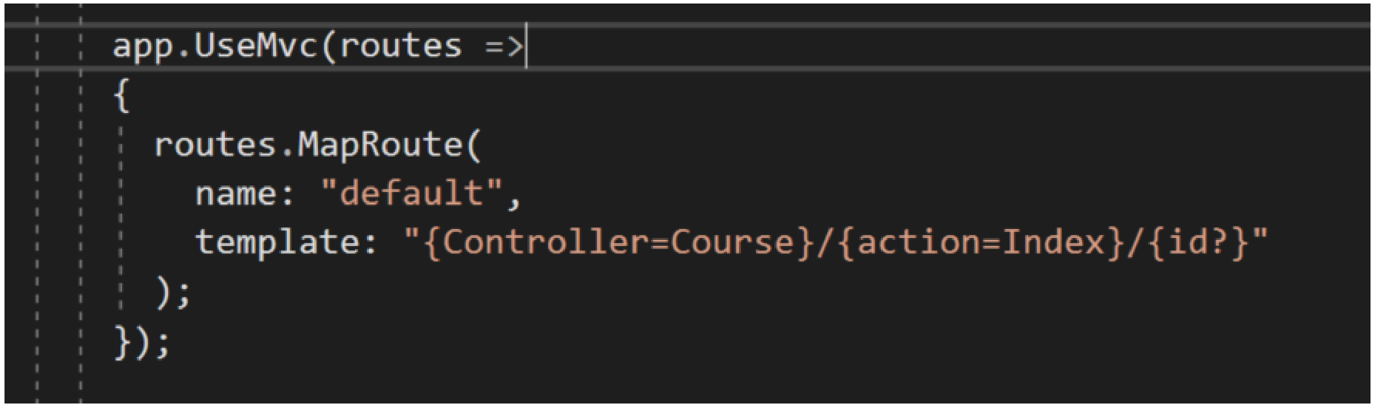
- CheckBox
- DropDownList
- EditTextBox
- Hidden
- ListBox
- Password
- RadioButton
- TextArea
- TextBox

Adding a View

- Right-click on View folder select Add View and configure view from the resulting dialog.
  - It’s easy to generate tables and lists that can be edited and posted back to the controller to effect changes to its model.
  - The HTML helpers on the previous page make building a view a fairly simple process.
  - The wizard for Strongly Typed views does most of the work in rendering model details.

What is a Controller?

- A controller is a C# class that derives from the class Controller.
    - A controller defines some category of processing for the application.
    - Its methods define the processing details.
    - Routing to a controller is defined in Startup.Configure method.



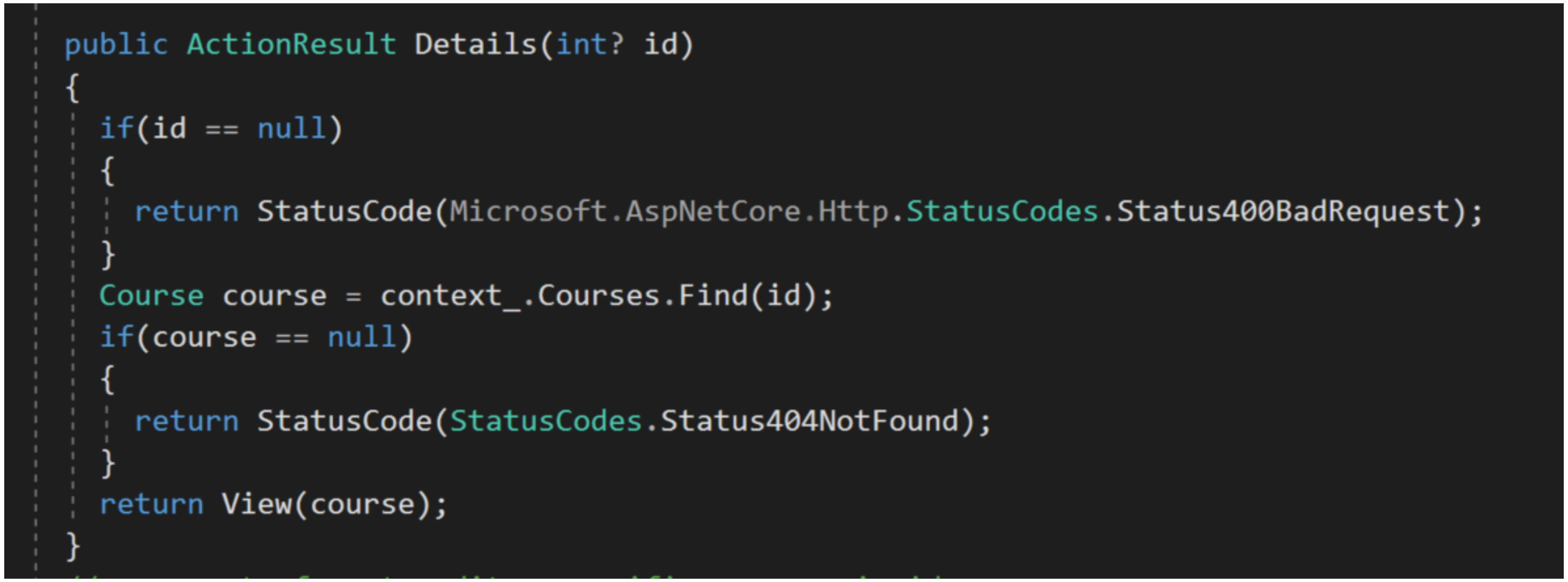
```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{Controller=Course}/{action=Index}/{id?}"
    );
});
```

Data Binding

- If a controller method takes a model class as a parameter, then the MVC infrastructure will instantiate an instance and pass to the controller method when requested via a url.
- On postback, if View parameters have the same names as model names, then the MVC infrastructure uses reflection to bind current view values to the model.

MvcSkeleton with CRUD Controller

- Action methods



```
public ActionResult Details(int? id)
{
    if(id == null)
    {
        return StatusCode(Microsoft.AspNetCore.Http.StatusCodes.Status400BadRequest);
    }
    Course course = context_.Courses.Find(id);
    if(course == null)
    {
        return StatusCode(StatusCodes.Status404NotFound);
    }
    return View(course);
}
```

Action returns ActionResult

- ActionResult: base class
- ContentResult: user defined object to Response
- EmptyResult: Nothing to Response
- FileResult: Send binary file to Response
- RedirectResult: redirect to url
- RedirectToRouteResult: redirect using routes
- JasonResult: send json to Response
- JavaScriptResult: send Javascript to Response
- ViewResult: Render a view

Adding a Controller

- Right-click on the Controller folder and select Add Controller.
  - Populate controller with methods whose names will become views and that take model parameters to supply views with data and react on postback to data changes made in view.

Web Application Development

- Create a new Asp.Net Core MVC project
    - Delete any part of that you don’t need
- Add a controller for each category of processing in your application:
    - A category is usually a few pages and db tables that focus on some particular application area
- Add methods to each controller for each request you wish to handle.
- Add views as needed for each controller action
- Add Model classes to support the application area:
    - Each model class has public properties that are synchronized with data in the model db or XML file.

An Opinion

- This Asp.Net Core MVC structure is very flexible:
  - You can have as many application categories as you need, simply by adding controllers.
  - The controllers keep the application well organized.
  - You can have as many views as you need. The navigation is simple and provided mostly by the MVC infrastructure, e.g., routing.
  - You can have as many models as you need. Just add classes and use Linq to access the data.

Things you may use

- LINQ – Language integrated query
  - Linq to XML and Linq to SQL are commonly used by models to provide data needed by a controller for one of its views.

That’s All Folks