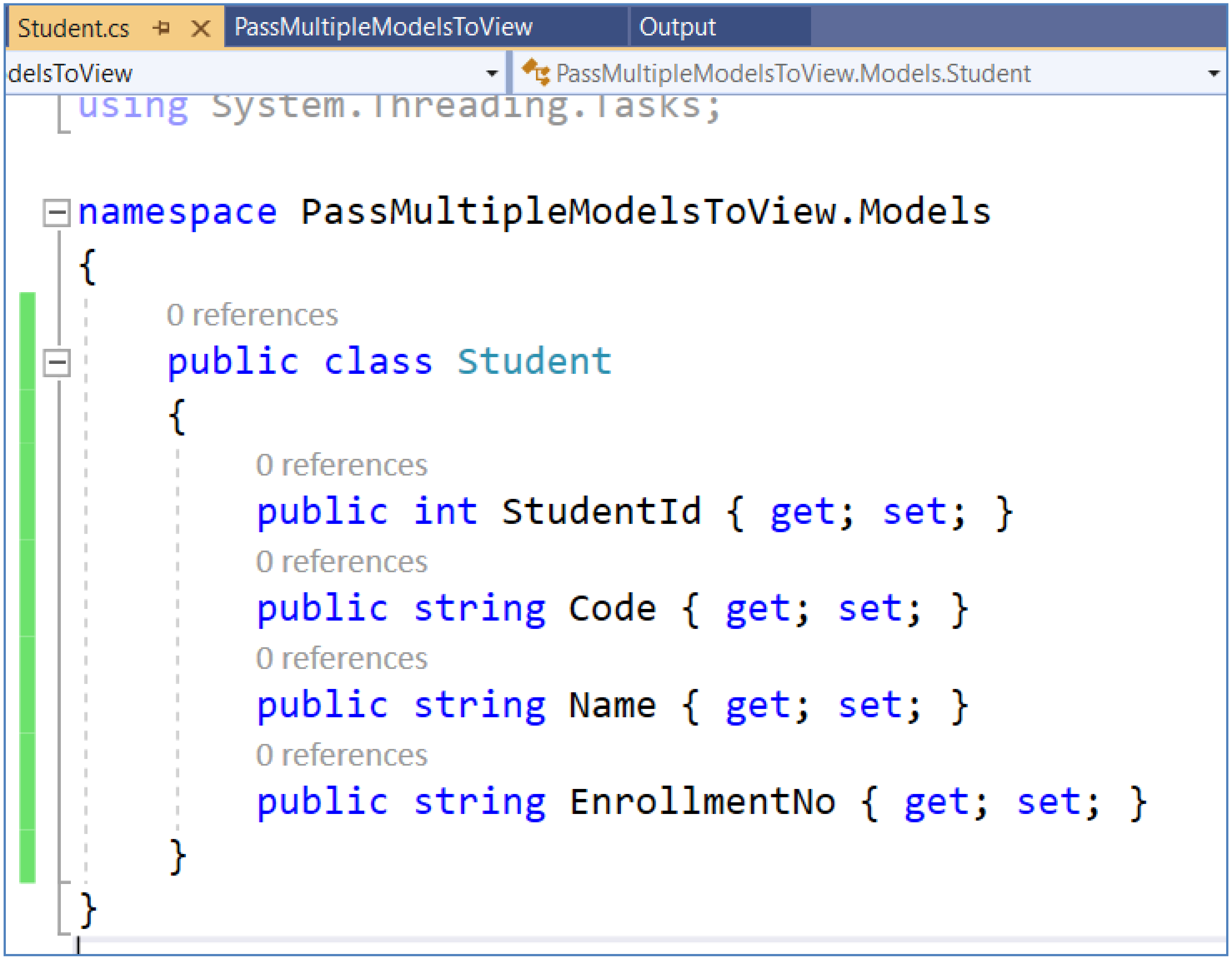
MVC Miscellaneous

Prepared for Vth semester DDU-CE students 2025-26 WAD

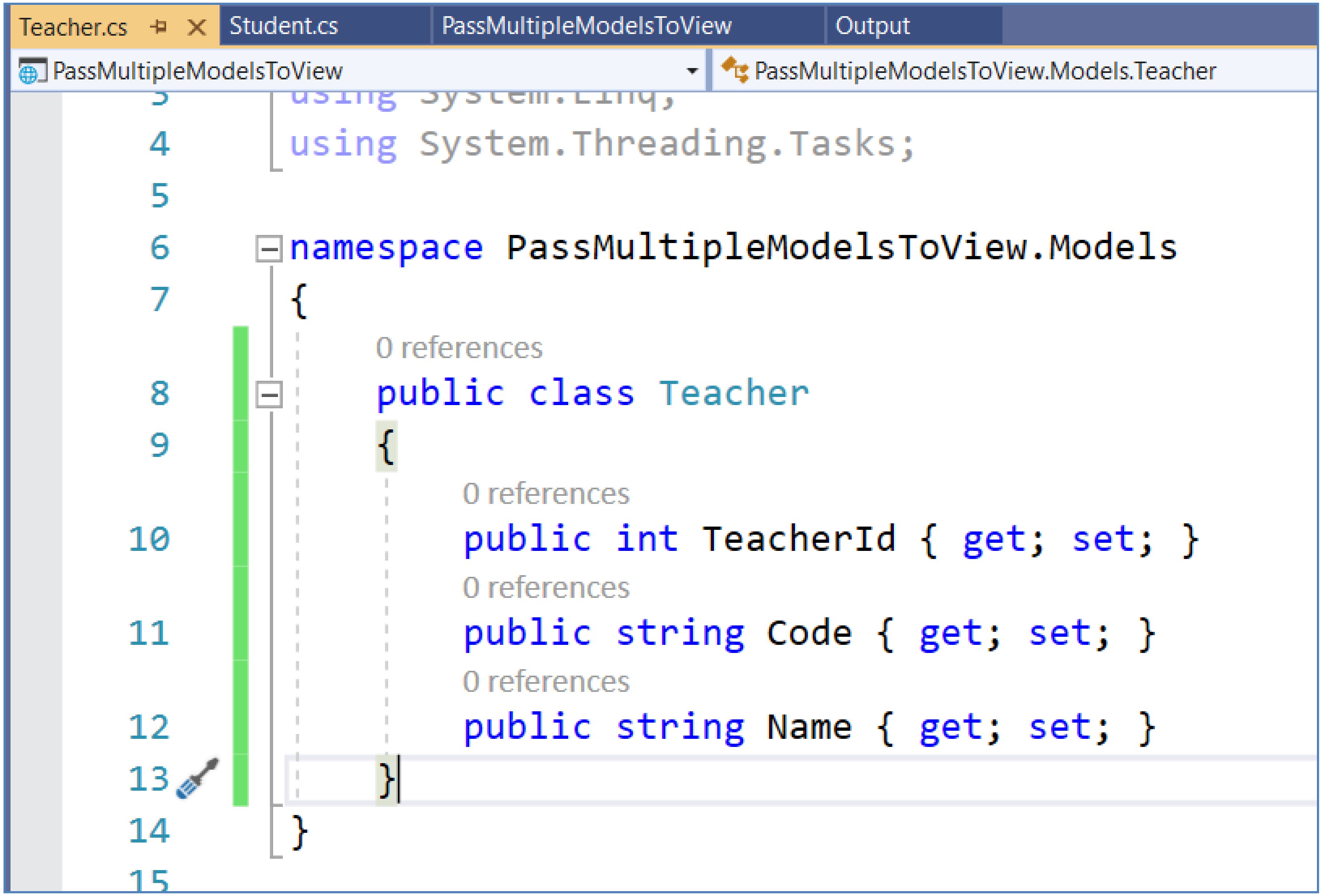
Apurva A Mehta

Multiple Models in Single View

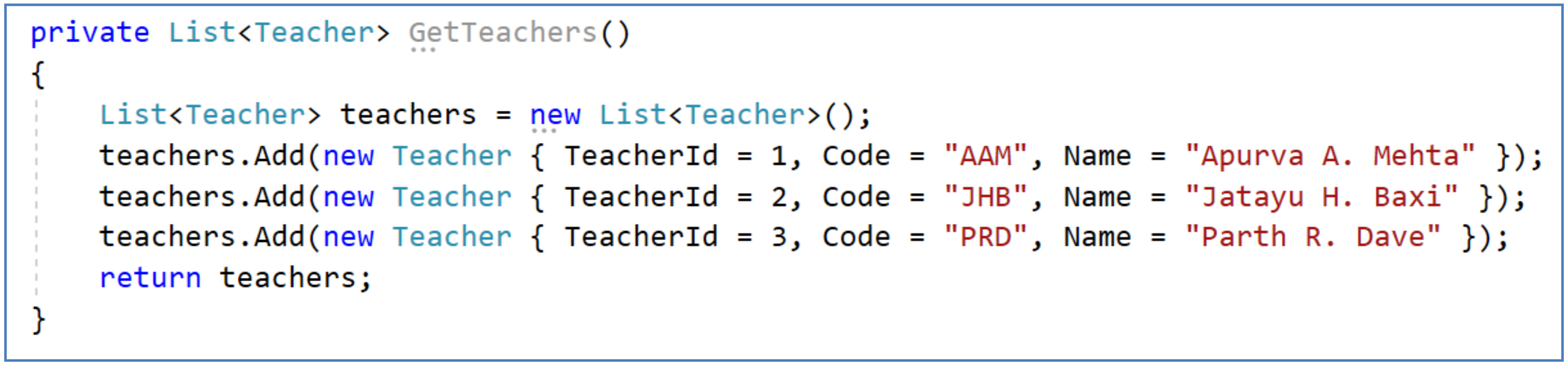
- Suppose We have two models, Teacher and Student, and We want to display a list of teachers and students within a single view.
- How can we do this?



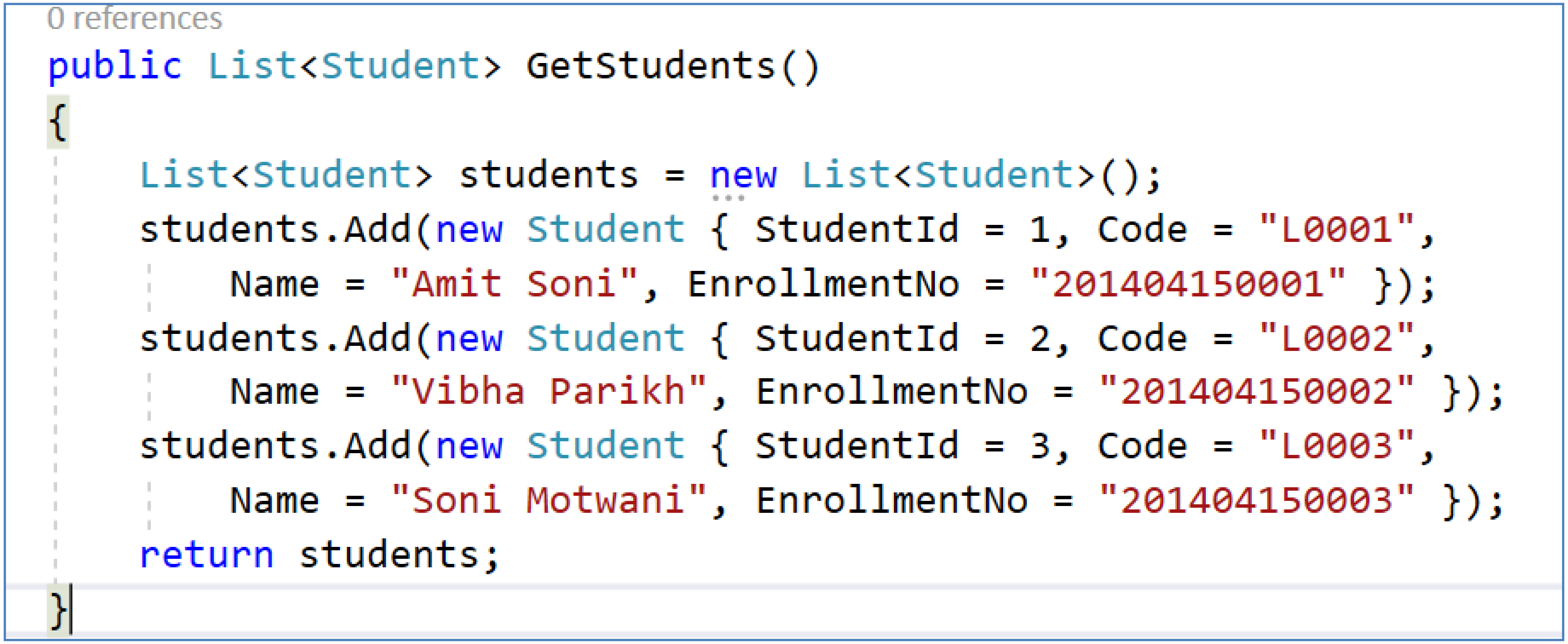
```
using System.Threading.Tasks;

namespace PassMultipleModelsToView.Models
{
    0 references
    public class Student
    {
        0 references
        public int StudentId { get; set; }
        0 references
        public string Code { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public string EnrollmentNo { get; set; }
    }
}
```



```
using System.Threading.Tasks;

namespace PassMultipleModelsToView.Models
{
    0 references
    public class Teacher
    {
        0 references
        public int TeacherId { get; set; }
        0 references
        public string Code { get; set; }
        0 references
        public string Name { get; set; }
    }
}
```

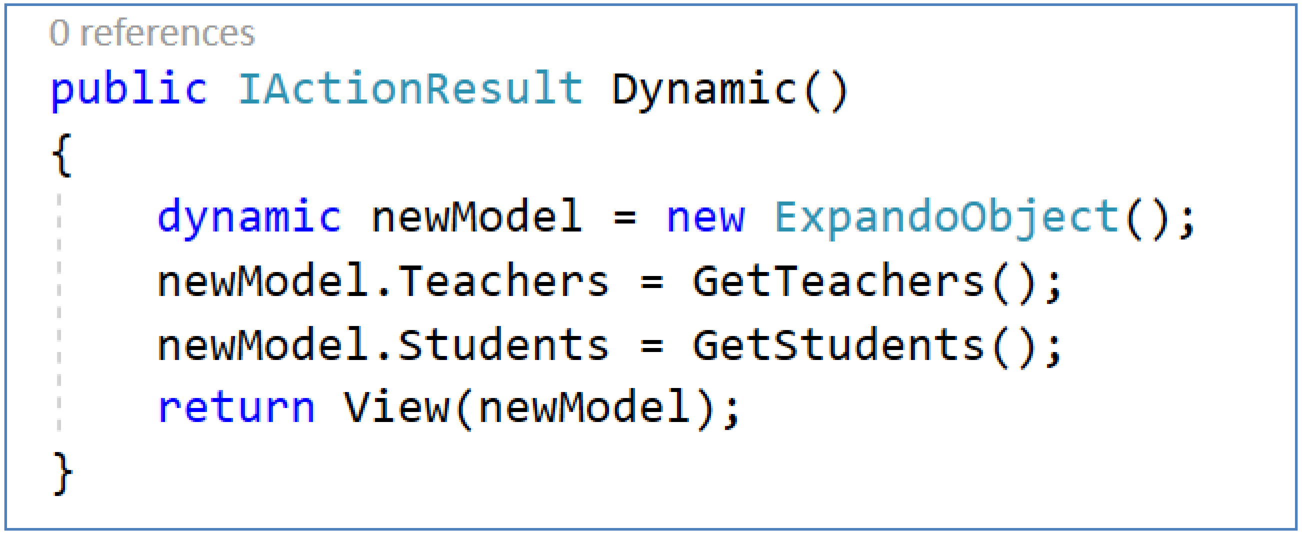


```
private List<Teacher> GetTeachers()
{
    List<Teacher> teachers = new List<Teacher>();
    teachers.Add(new Teacher { TeacherId = 1, Code = "AAM", Name = "Apurva A. Mehta" });
    teachers.Add(new Teacher { TeacherId = 2, Code = "JHB", Name = "Jatayu H. Baxi" });
    teachers.Add(new Teacher { TeacherId = 3, Code = "PRD", Name = "Parth R. Dave" });
    return teachers;
}
```

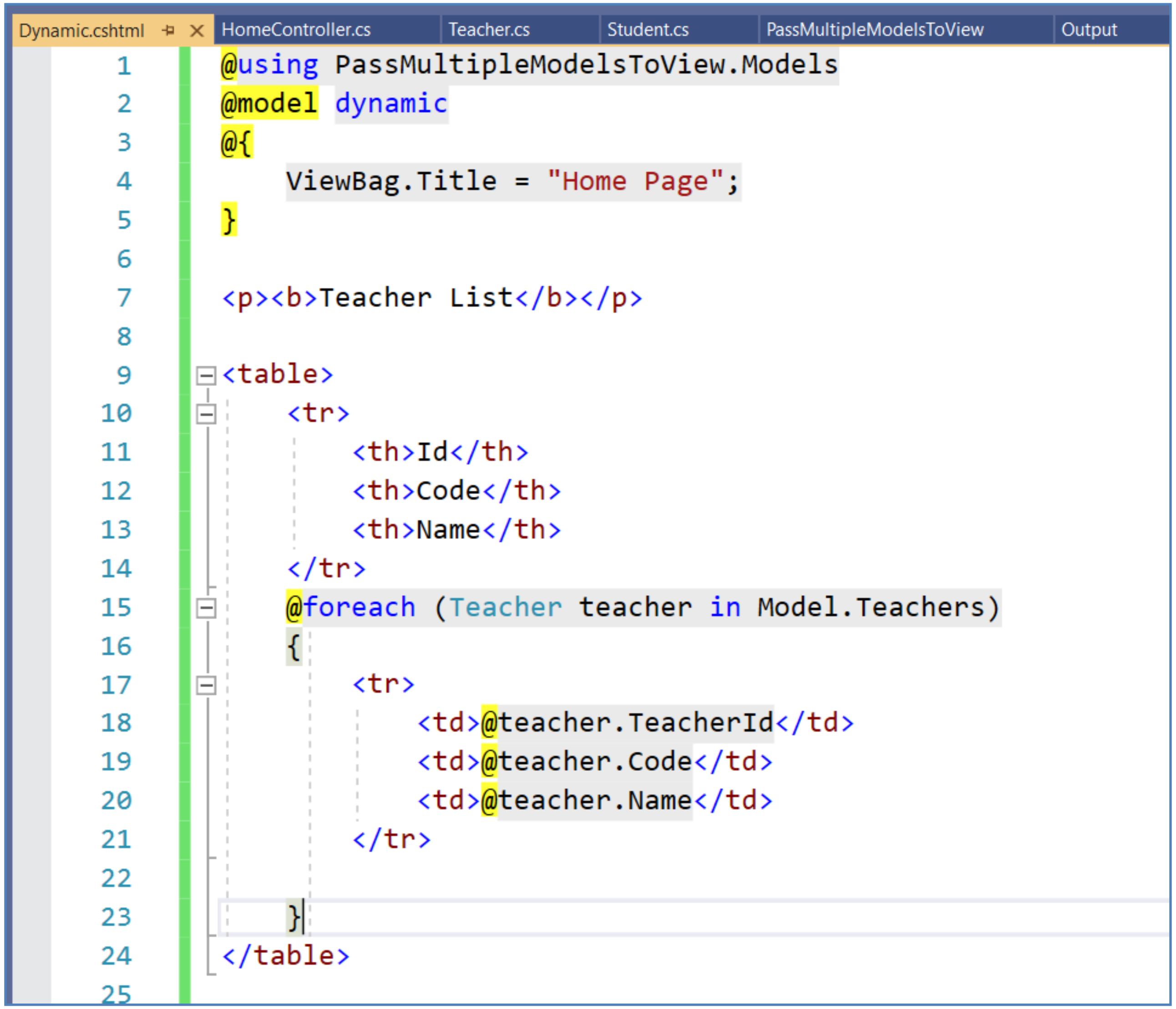


```
0 references
public List<Student> GetStudents()
{
    List<Student> students = new List<Student>();
    students.Add(new Student { StudentId = 1, Code = "L0001",
        Name = "Amit Soni", EnrollmentNo = "201404150001" });
    students.Add(new Student { StudentId = 2, Code = "L0002",
        Name = "Vibha Parikh", EnrollmentNo = "201404150002" });
    students.Add(new Student { StudentId = 3, Code = "L0003",
        Name = "Soni Motwani", EnrollmentNo = "201404150003" });
    return students;
}
```

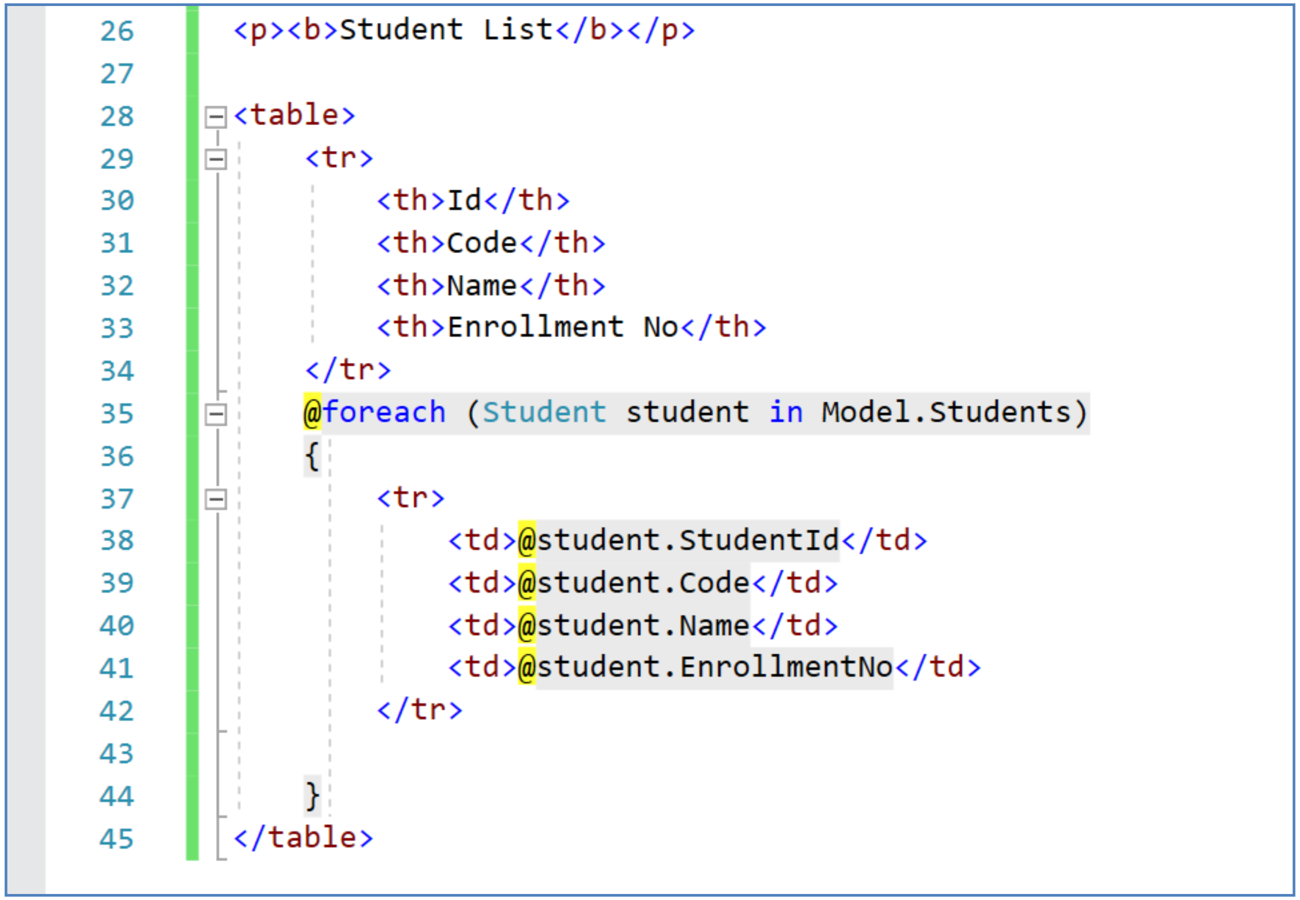
1. Using Dynamic Model



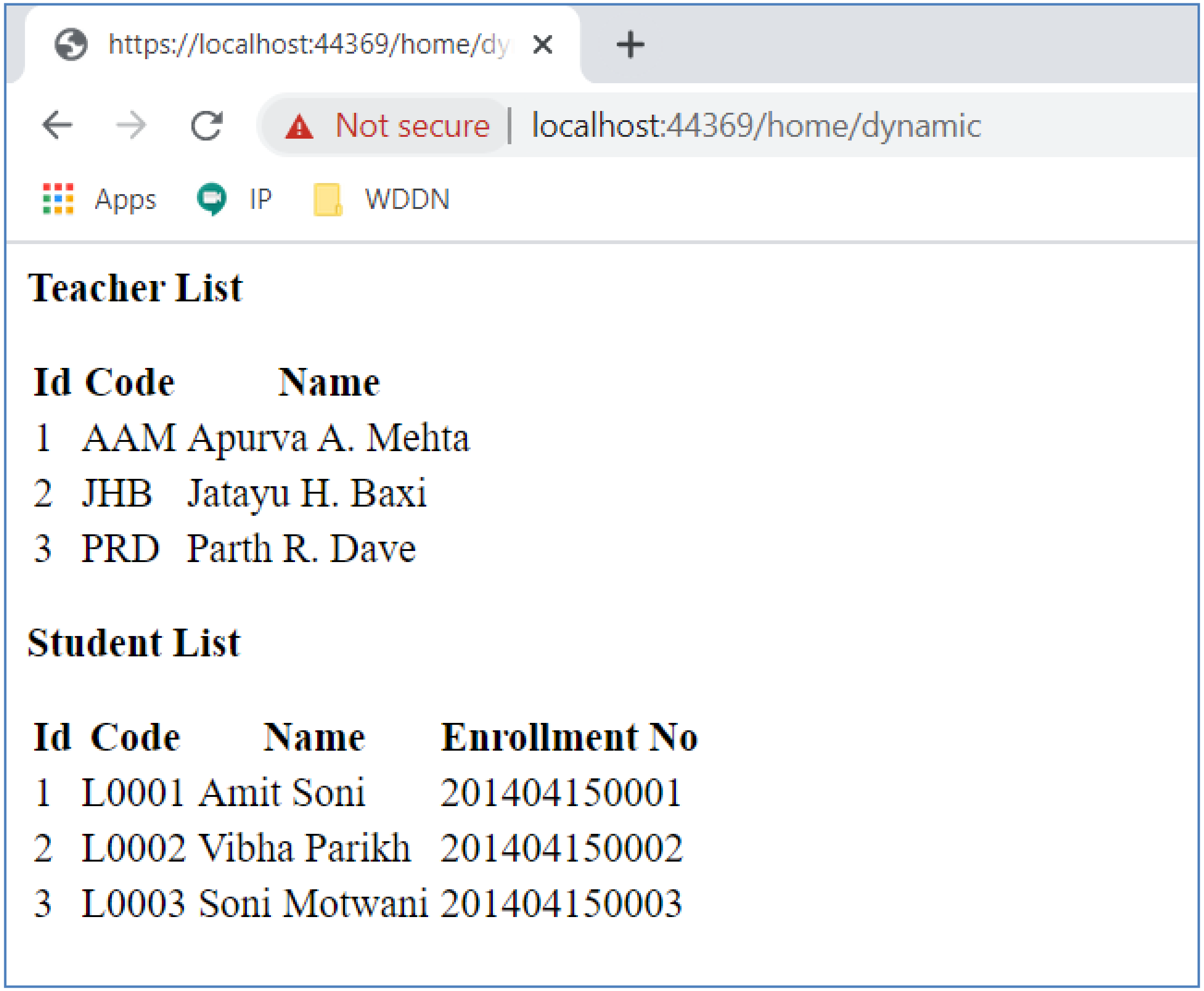
```
0 references
public IActionResult Dynamic()
{
    dynamic newModel = new ExpandoObject();
    newModel.Teachers = GetTeachers();
    newModel.Students = GetStudents();
    return View(newModel);
}
```



```
@using PassMultipleModelsToView.Models
@model dynamic
@{
    ViewBag.Title = "Home Page";
}

<p><b>Teacher List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
    </tr>
    @foreach (Teacher teacher in Model.Teachers)
    {
        <tr>
            <td>@teacher.TeacherId</td>
            <td>@teacher.Code</td>
            <td>@teacher.Name</td>
        </tr>

    }
</table>
```



```
<p><b>Student List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
        <th>Enrollment No</th>
    </tr>
    @foreach (Student student in Model.Students)
    {
        <tr>
            <td>@student.StudentId</td>
            <td>@student.Code</td>
            <td>@student.Name</td>
            <td>@student.EnrollmentNo</td>
        </tr>

    }
</table>
```



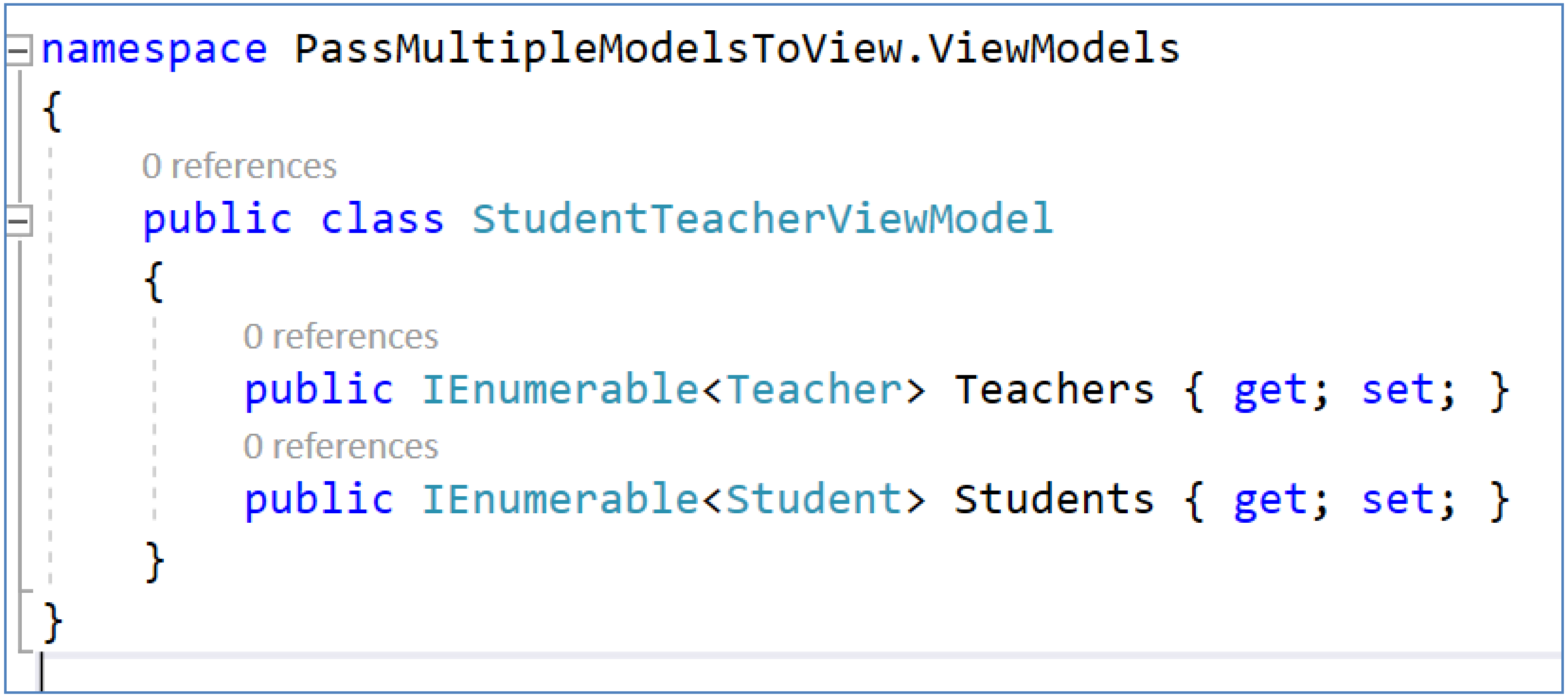
Teacher List

Id	Code	Name
1	AAM	Apurva A. Mehta
2	JHB	Jatayu H. Baxi
3	PRD	Parth R. Dave

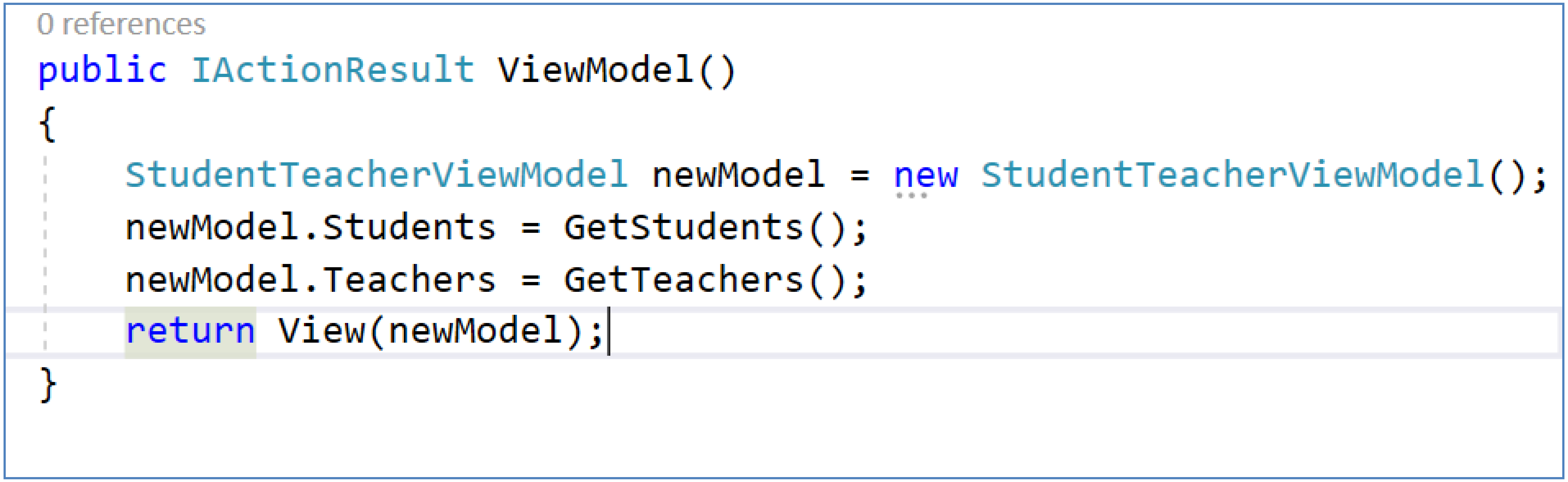
Student List

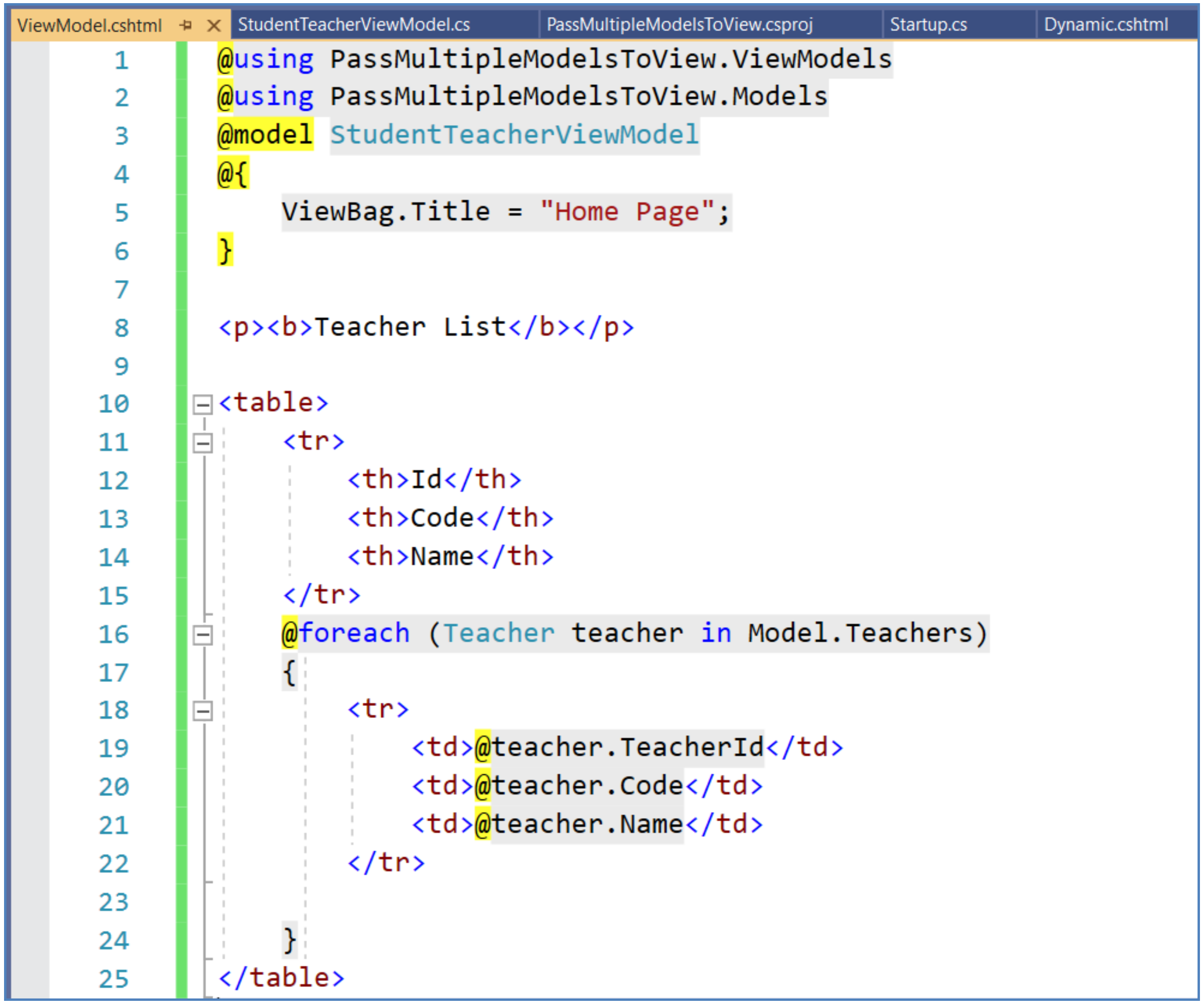
Id	Code	Name	Enrollment No
1	L0001	Amit Soni	201404150001
2	L0002	Vibha Parikh	201404150002
3	L0003	Soni Motwani	201404150003

2. Using View Model

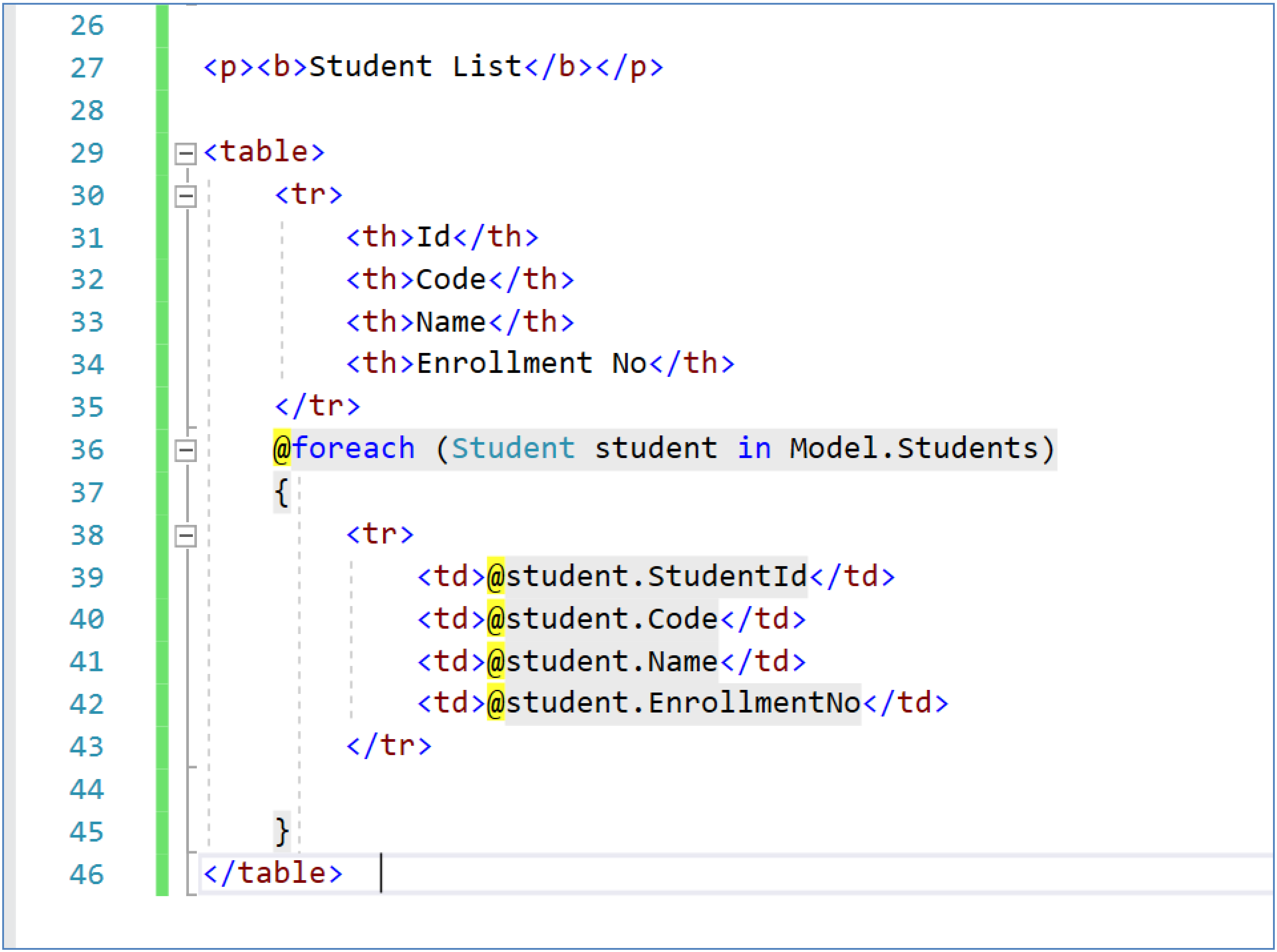


```
namespace PassMultipleModelsToView.ViewModels
{
        0 references
        public class StudentTeacherViewModel
        {
                0 references
                public IEnumerable<Teacher> Teachers { get; set; }
                0 references
                public IEnumerable<Student> Students { get; set; }
        }
}
```

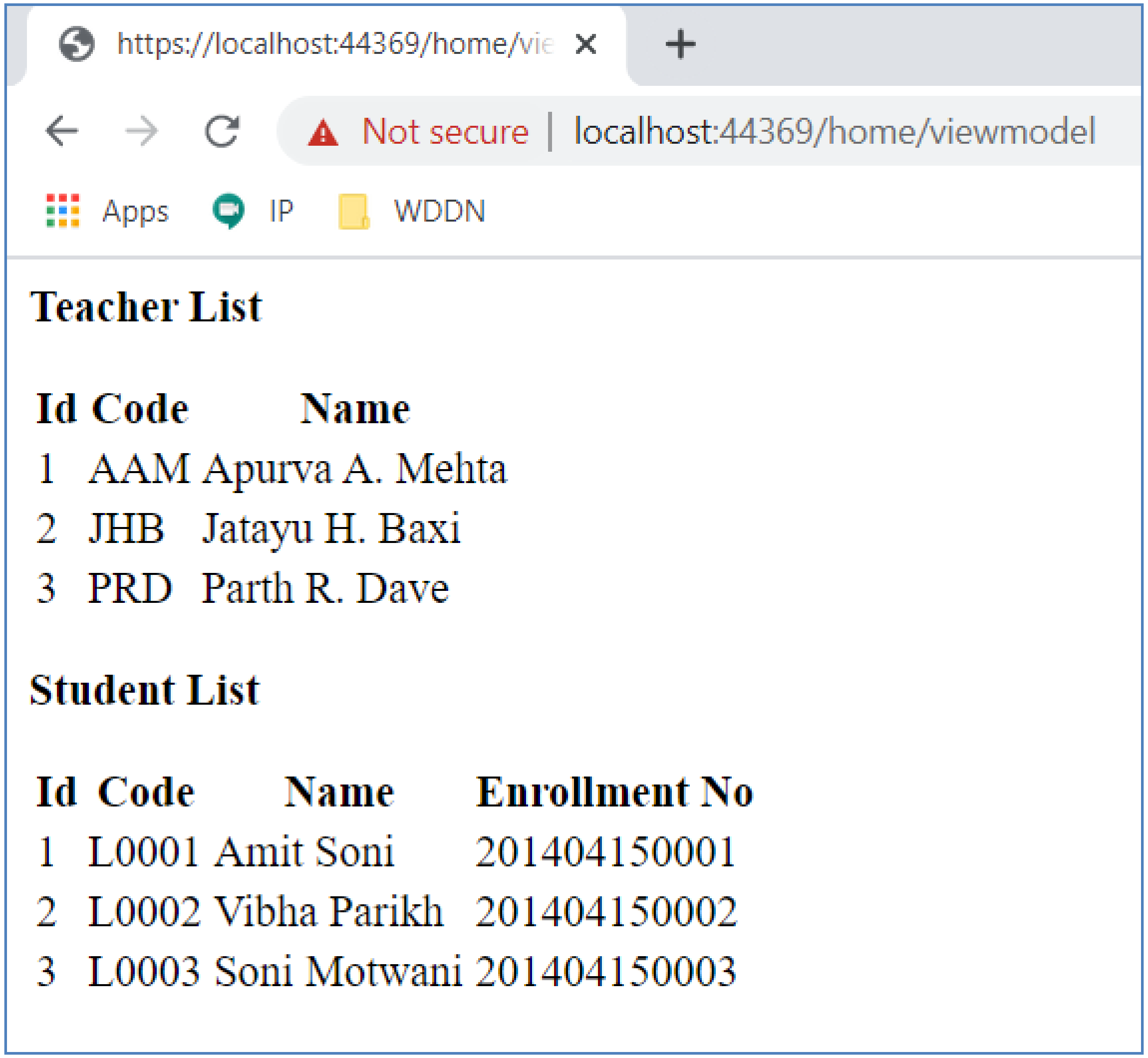

0 references
```
public IActionResult ViewModel()
{
    StudentTeacherViewModel newModel = new StudentTeacherViewModel();
    newModel.Students = GetStudents();
    newModel.Teachers = GetTeachers();
    return View(newModel);
}
```


ViewModel.cshtml | StudentTeacherViewModel.cs | PassMultipleModelsToView.csproj | Startup.cs | Dynamic.cshtml

```
@using PassMultipleModelsToView.ViewModels
@using PassMultipleModelsToView.Models
@model StudentTeacherViewModel
@{
    ViewBag.Title = "Home Page";
}

<p><b>Teacher List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
    </tr>
    @foreach (Teacher teacher in Model.Teachers)
    {
        <tr>
            <td>@teacher.TeacherId</td>
            <td>@teacher.Code</td>
            <td>@teacher.Name</td>
        </tr>

    }
</table>
```



```
<p><b>Student List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
        <th>Enrollment No</th>
    </tr>
    @foreach (Student student in Model.Students)
    {
        <tr>
            <td>@student.StudentId</td>
            <td>@student.Code</td>
            <td>@student.Name</td>
            <td>@student.EnrollmentNo</td>
        </tr>

    }
</table>
```


https://localhost:44369/home/vie
Not secure
localhost:44369/home/viewmodel
Apps
IP
WDDN

Teacher List

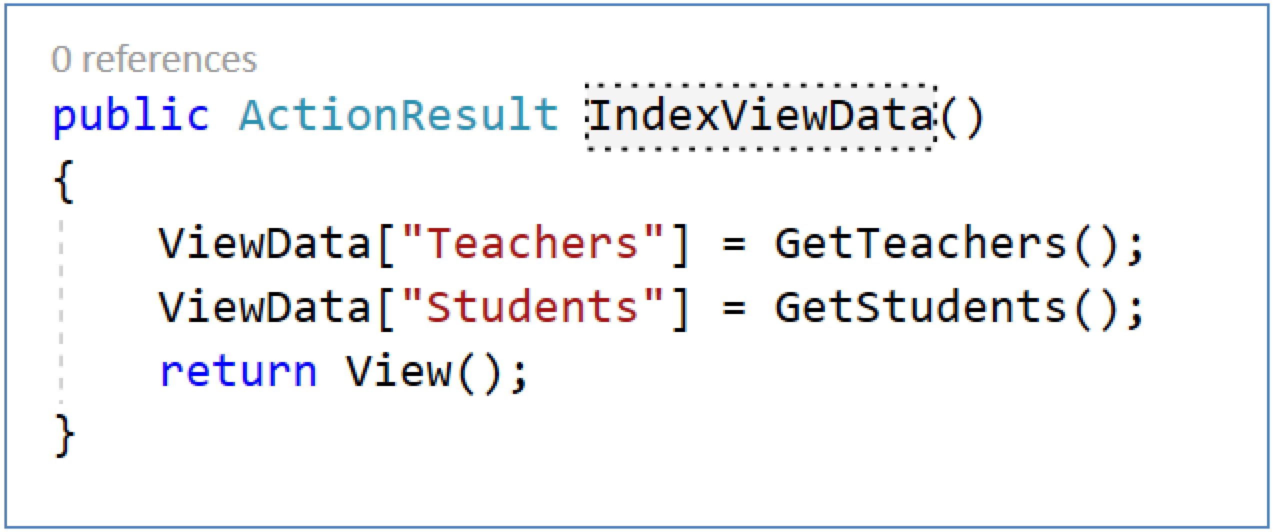
Id	Code	Name
1	AAM	Apurva A. Mehta
2	JHB	Jatayu H. Baxi
3	PRD	Parth R. Dave

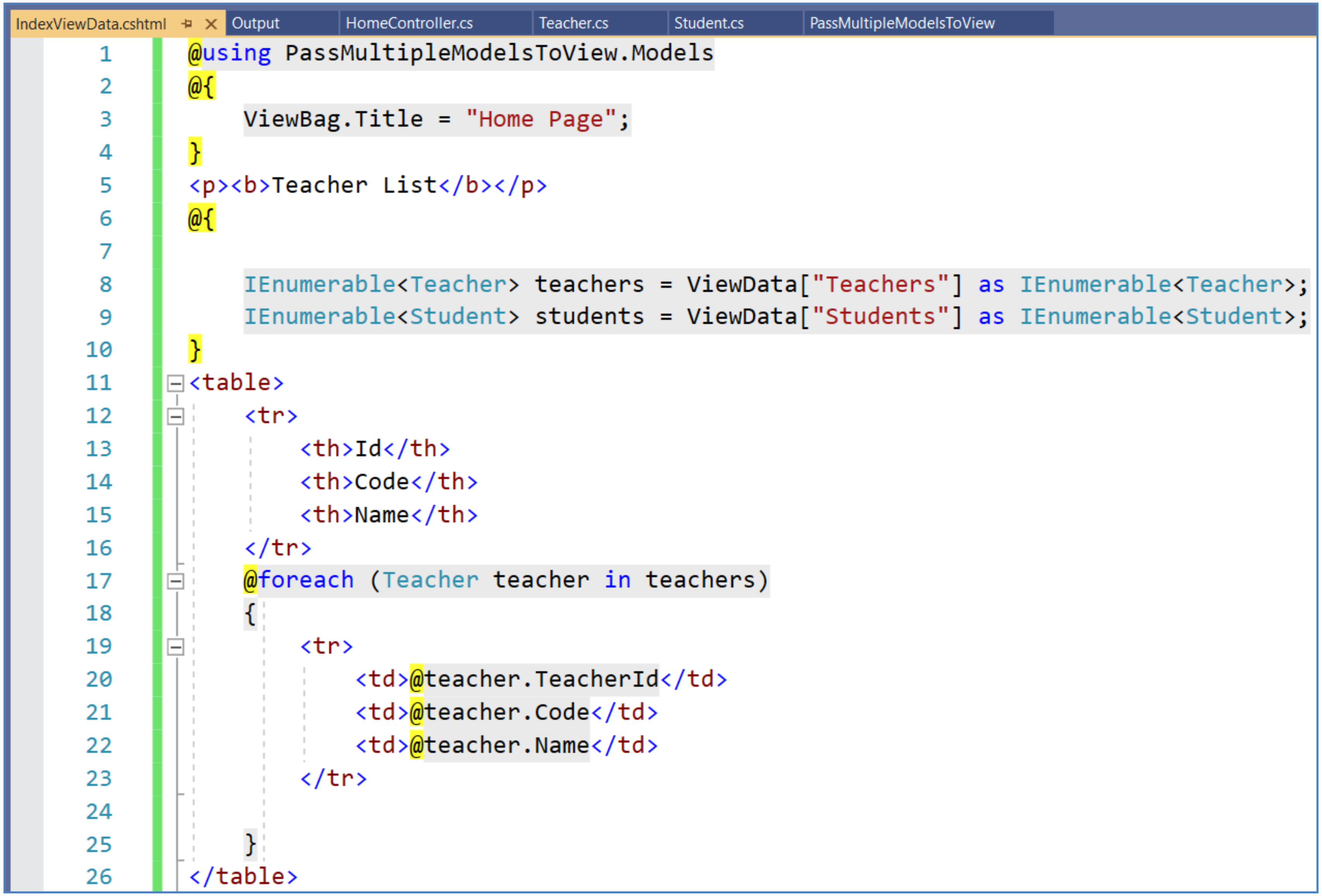
Student List

Id	Code	Name	Enrollment No
1	L0001	Amit Soni	201404150001
2	L0002	Vibha Parikh	201404150002
3	L0003	Soni Motwani	201404150003

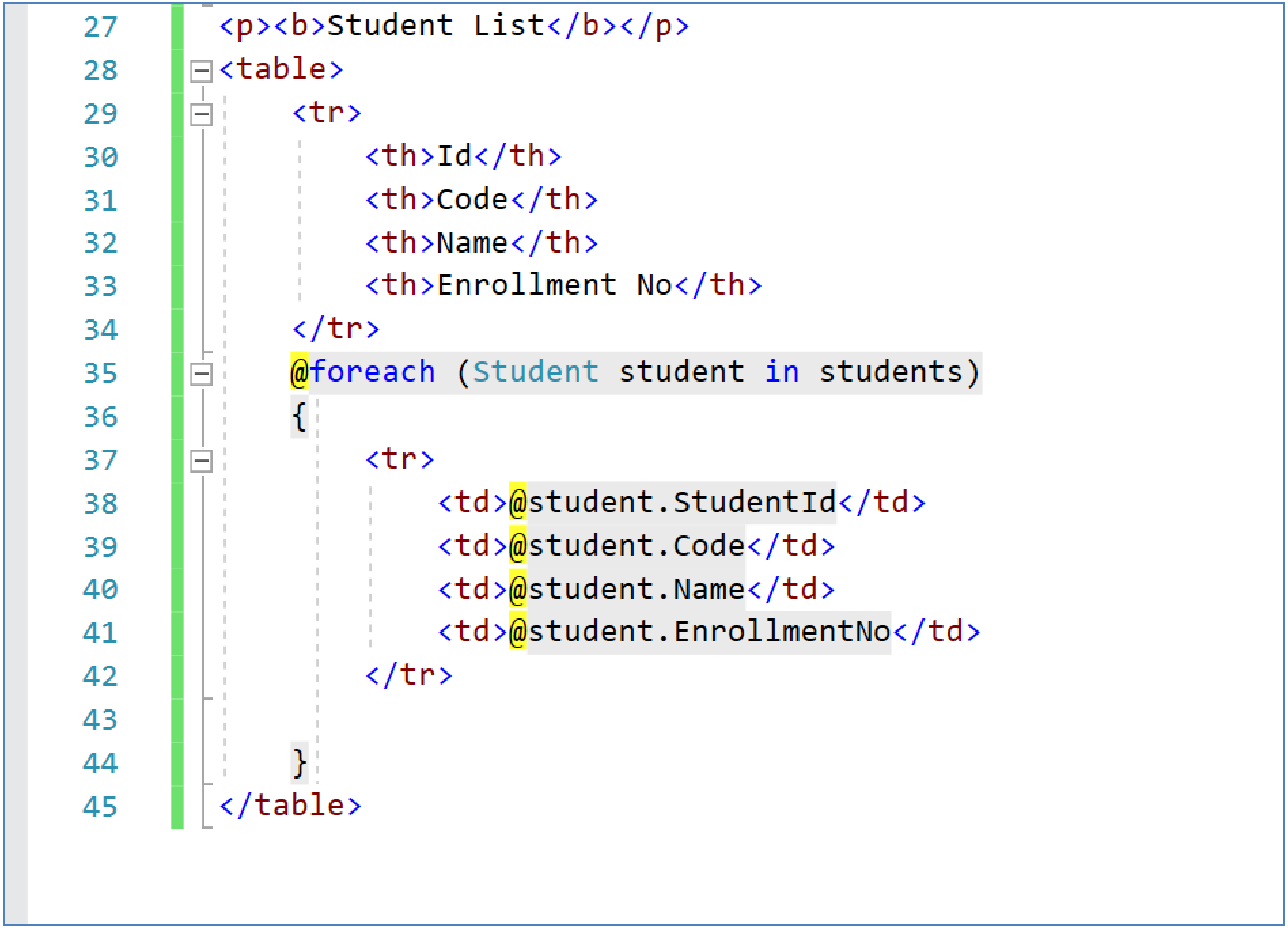
ViewData and ViewBag

3. Using ViewData

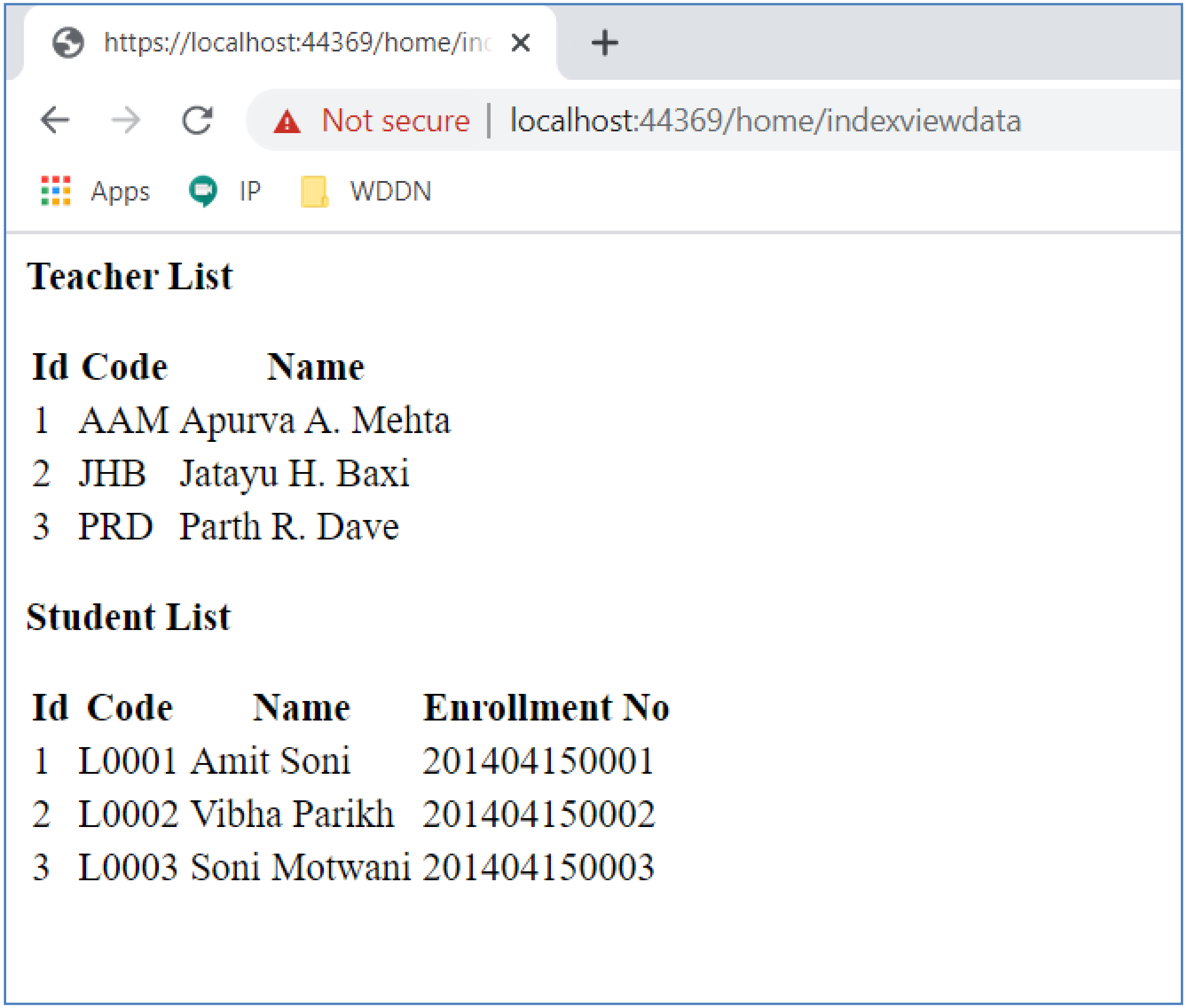

```
0 references
public ActionResult IndexViewData()
{
    ViewData["Teachers"] = GetTeachers();
    ViewData["Students"] = GetStudents();
    return View();
}
```



```
@using PassMultipleModelsToView.Models
@{
    ViewBag.Title = "Home Page";
}
<p><b>Teacher List</b></p>
@{

    IEnumerable<Teacher> teachers = ViewData["Teachers"] as IEnumerable<Teacher>;
    IEnumerable<Student> students = ViewData["Students"] as IEnumerable<Student>;
}
<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
    </tr>
    @foreach (Teacher teacher in teachers)
    {
        <tr>
            <td>@teacher.TeacherId</td>
            <td>@teacher.Code</td>
            <td>@teacher.Name</td>
        </tr>

    }
</table>
```



```
<p><b>Student List</b></p>
<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
        <th>Enrollment No</th>
    </tr>
    @foreach (Student student in students)
    {
        <tr>
            <td>@student.StudentId</td>
            <td>@student.Code</td>
            <td>@student.Name</td>
            <td>@student.EnrollmentNo</td>
        </tr>

    }
</table>
```


https://localhost:44369/home/inc
Not secure | localhost:44369/home/indexviewdata
Apps
IP
WDDN

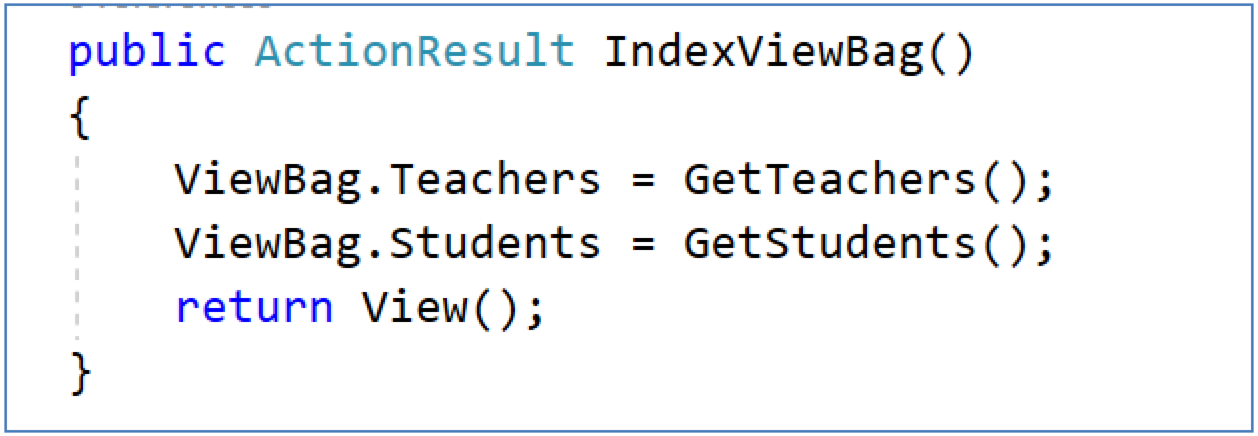
Teacher List

Id	Code	Name
1	AAM	Apurva A. Mehta
2	JHB	Jatayu H. Baxi
3	PRD	Parth R. Dave

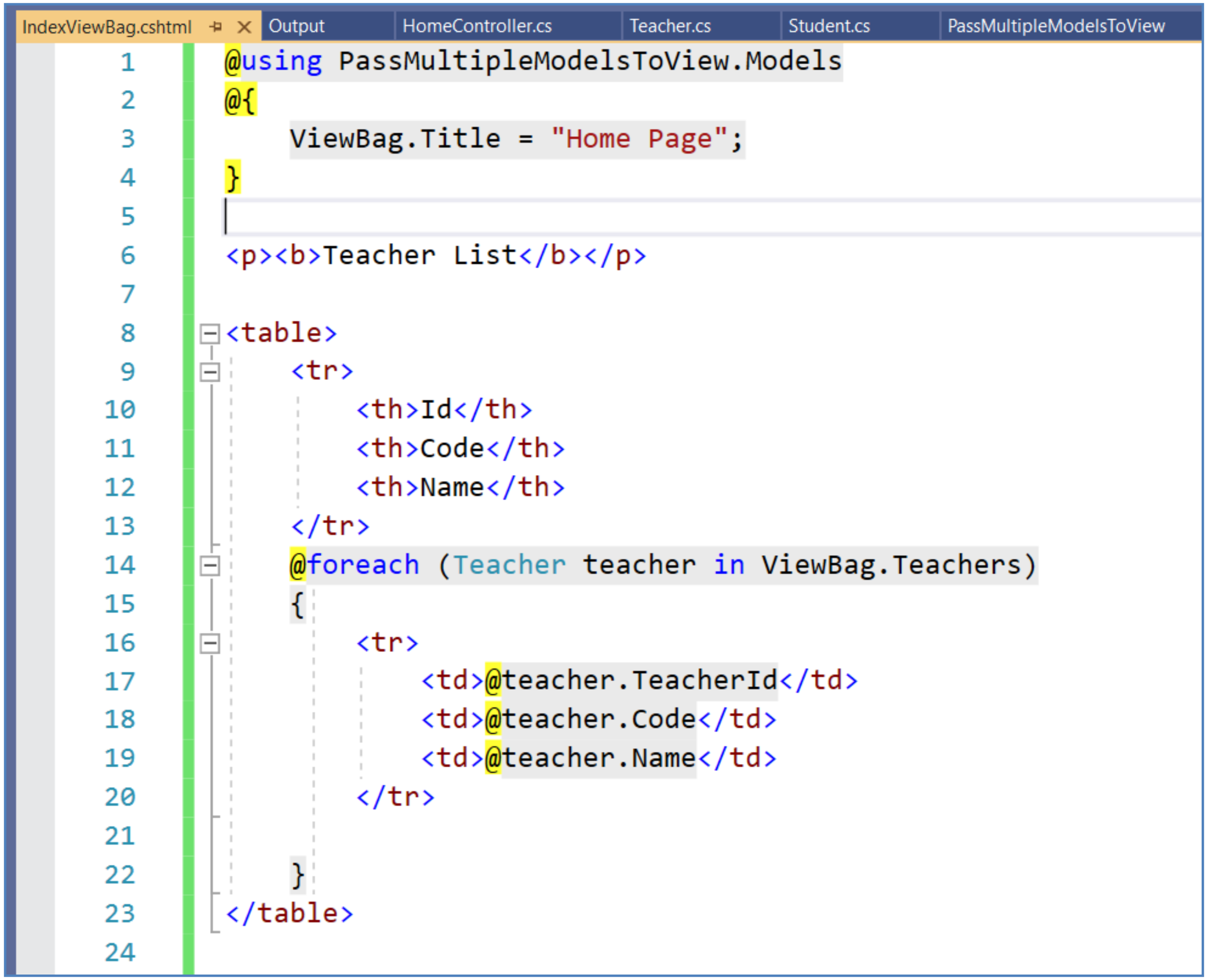
Student List

Id	Code	Name	Enrollment No
1	L0001	Amit Soni	201404150001
2	L0002	Vibha Parikh	201404150002
3	L0003	Soni Motwani	201404150003

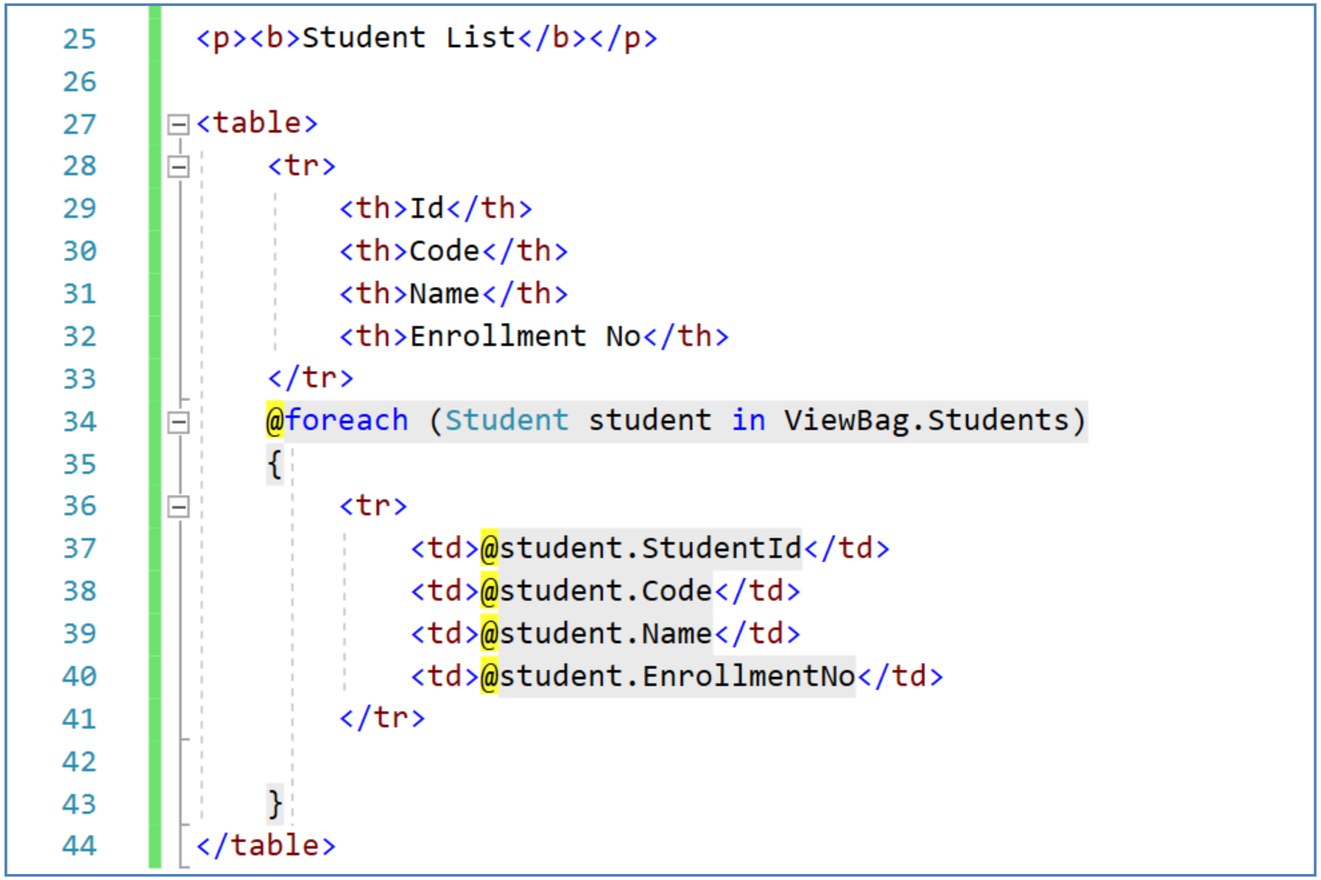
4. Using ViewBag



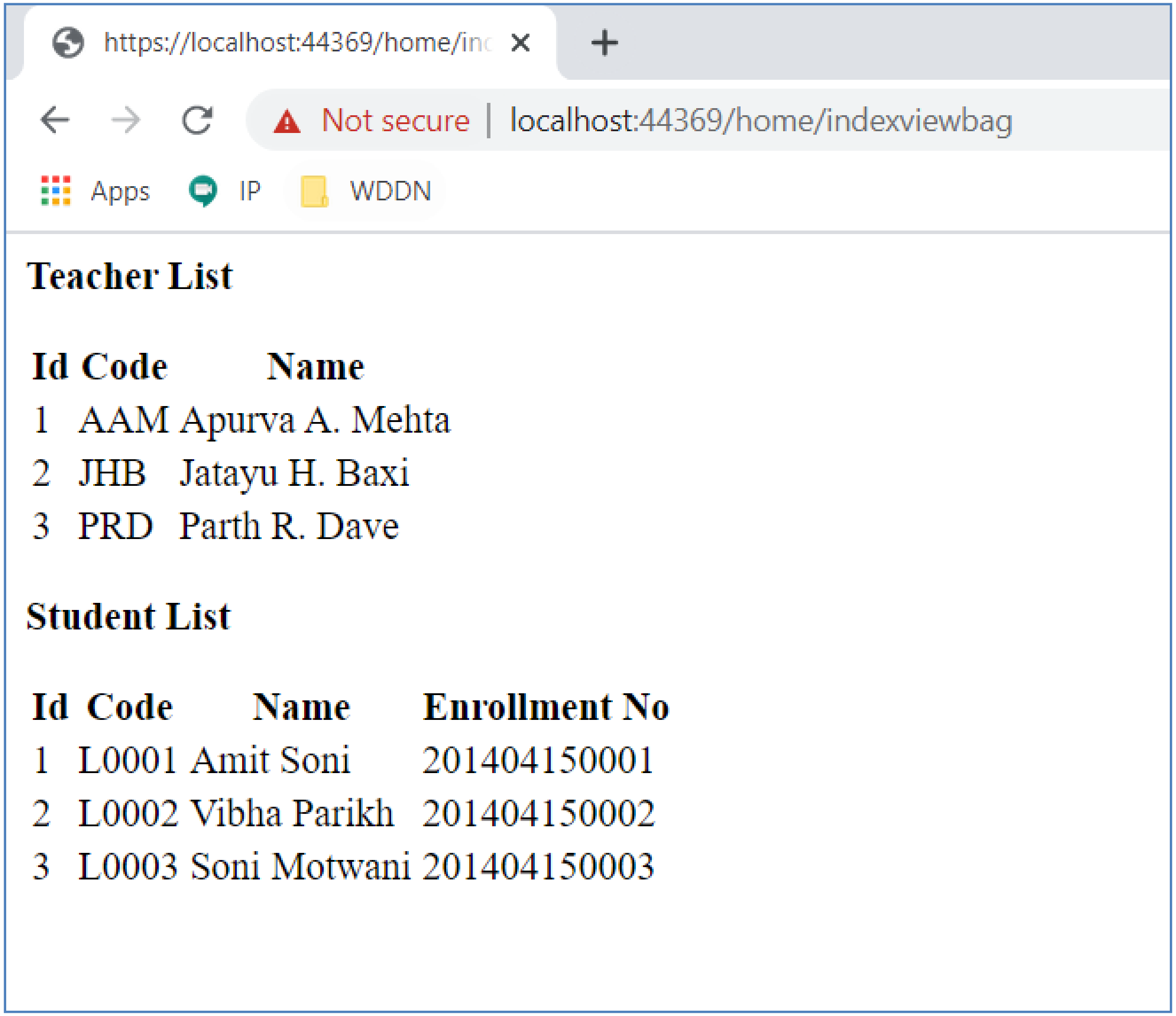
```
public ActionResult IndexViewBag()
{
    ViewBag.Teachers = GetTeachers();
    ViewBag.Students = GetStudents();
    return View();
}
```



```
@using PassMultipleModelsToView.Models
@{
    ViewBag.Title = "Home Page";
}

<p><b>Teacher List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
    </tr>
    @foreach (Teacher teacher in ViewBag.Teachers)
    {
        <tr>
            <td>@teacher.TeacherId</td>
            <td>@teacher.Code</td>
            <td>@teacher.Name</td>
        </tr>

    }
</table>

```



```
<p><b>Student List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
        <th>Enrollment No</th>
    </tr>
    @foreach (Student student in ViewBag.Students)
    {
        <tr>
            <td>@student.StudentId</td>
            <td>@student.Code</td>
            <td>@student.Name</td>
            <td>@student.EnrollmentNo</td>
        </tr>

    }
</table>
```


https://localhost:44369/home/inc
Not secure | localhost:44369/home/indexviewbag
Apps
IP
WDDN

Teacher List

Id	Code	Name
1	AAM	Apurva A. Mehta
2	JHB	Jatayu H. Baxi
3	PRD	Parth R. Dave

Student List

Id	Code	Name	Enrollment No
1	L0001	Amit Soni	201404150001
2	L0002	Vibha Parikh	201404150002
3	L0003	Soni Motwani	201404150003

View Components

- New to ASP.NET Core MVC, view components are similar to partial views, but they are much more powerful.
- View components don’t use model binding, and only depend on the data you provide when calling into it.

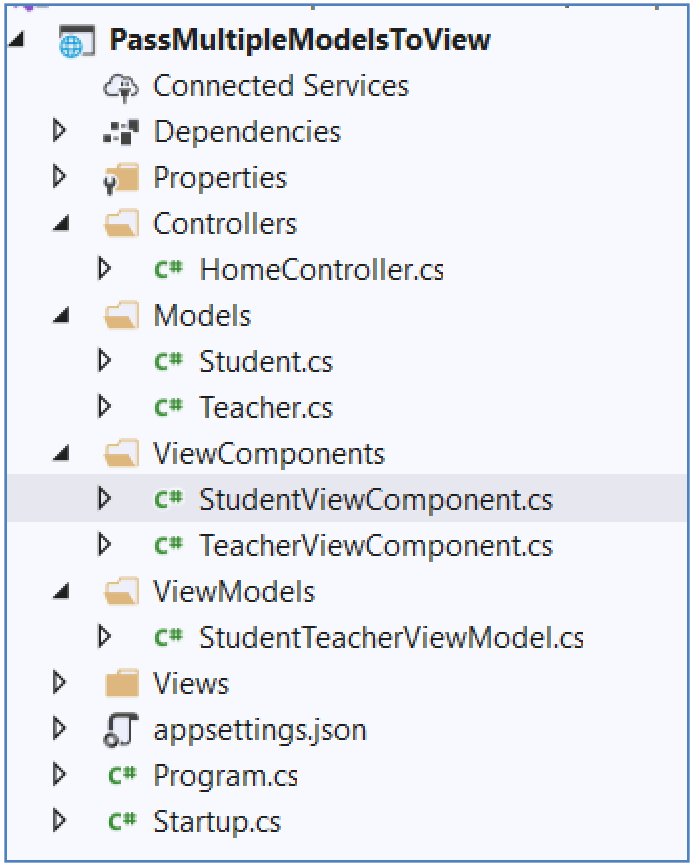
Cont.

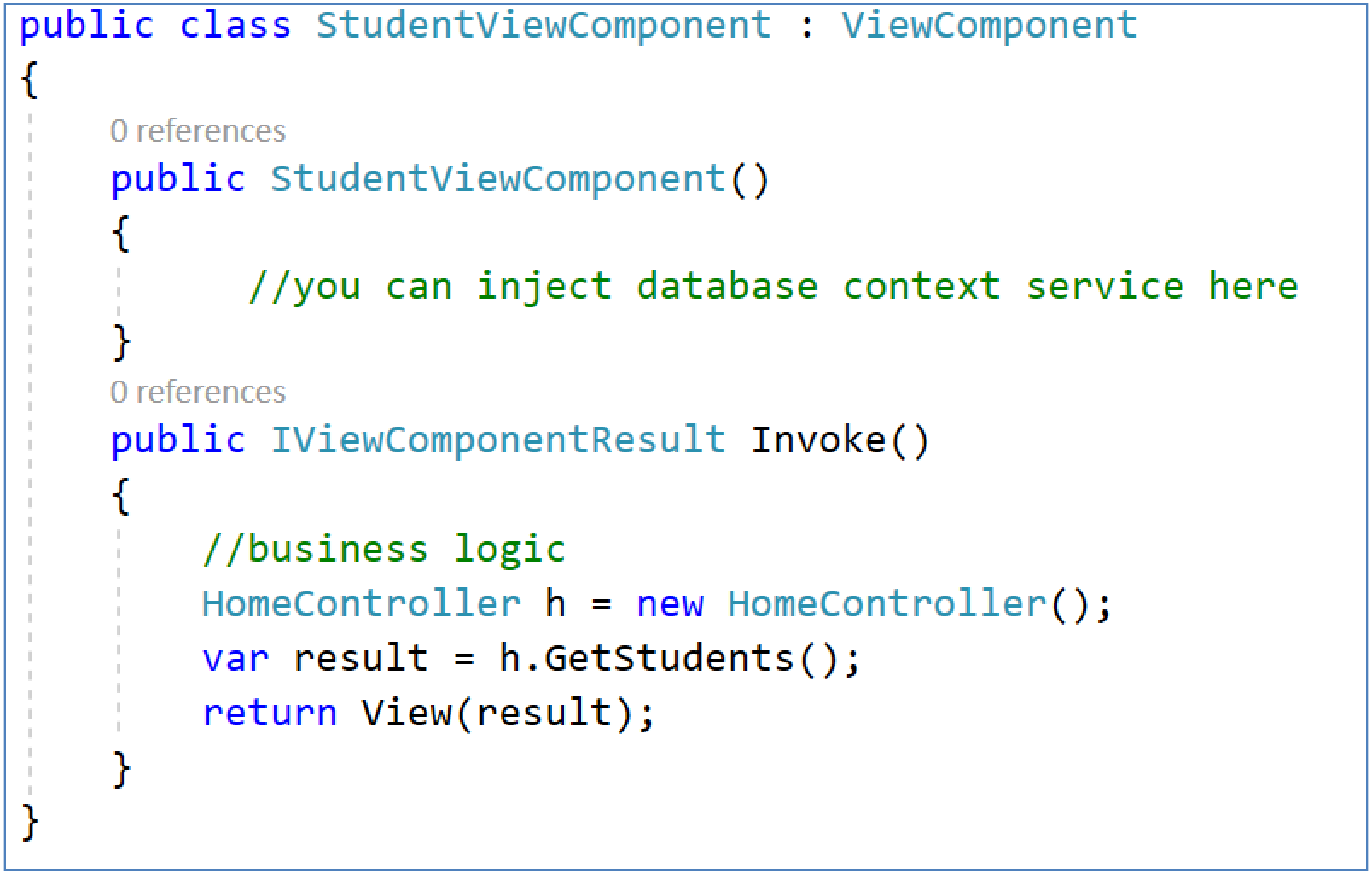
- A view component:
  - Renders a chunk rather than a whole response
  - Includes the same separation-of-concerns and testability benefits found between a controller and view
  - Can have parameters and business logic
  - Is typically invoked from a layout page

Cont.

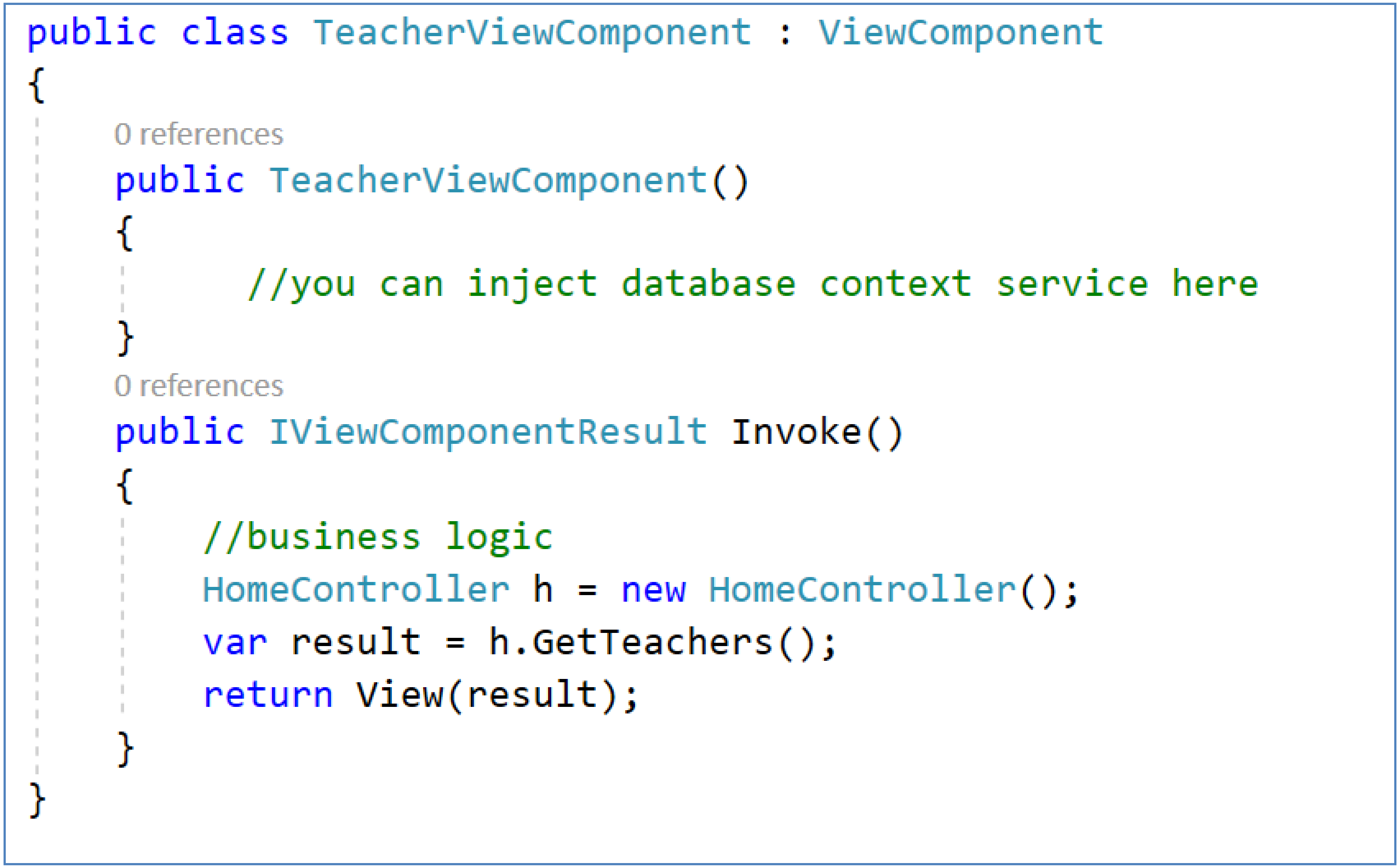
- A view component consists of two parts, the class (typically derived from ViewComponent) and the result it returns (typically a view).
- Like controllers, a view component can be a POCO, but most developers will want to take advantage of the methods and properties available by deriving from ViewComponent.

5. Using View Component

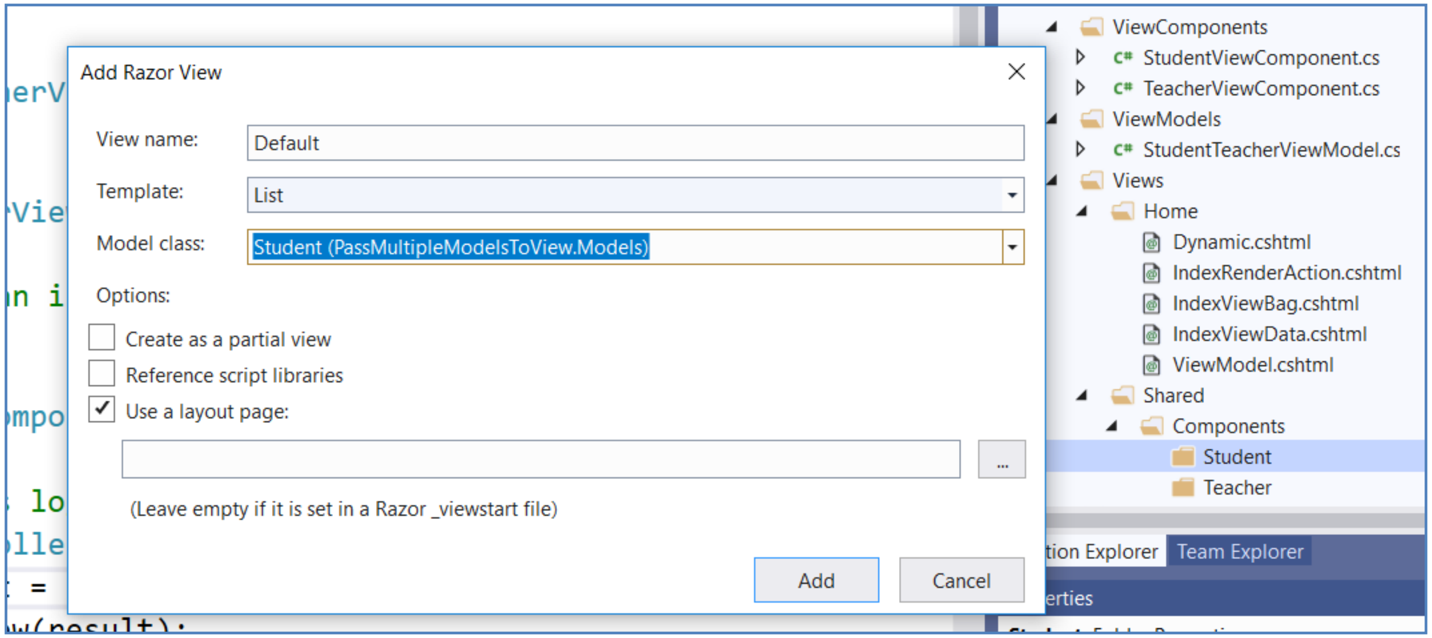




```
public class StudentViewComponent : ViewComponent
{
        0 references
        public StudentViewComponent()
        {
                //you can inject database context service here
        }
        0 references
        public IViewComponentResult Invoke()
        {
            //business logic
            HomeController h = new HomeController();
            var result = h.GetStudents();
            return View(result);
        }
}
```



```
public class TeacherViewComponent : ViewComponent
{
    0 references
    public TeacherViewComponent()
    {
            //you can inject database context service here
    }
    0 references
    public IViewComponentResult Invoke()
    {
        //business logic
        HomeController h = new HomeController();
        var result = h.GetTeachers();
        return View(result);
    }
}
```



Add Razor View

View name: Default

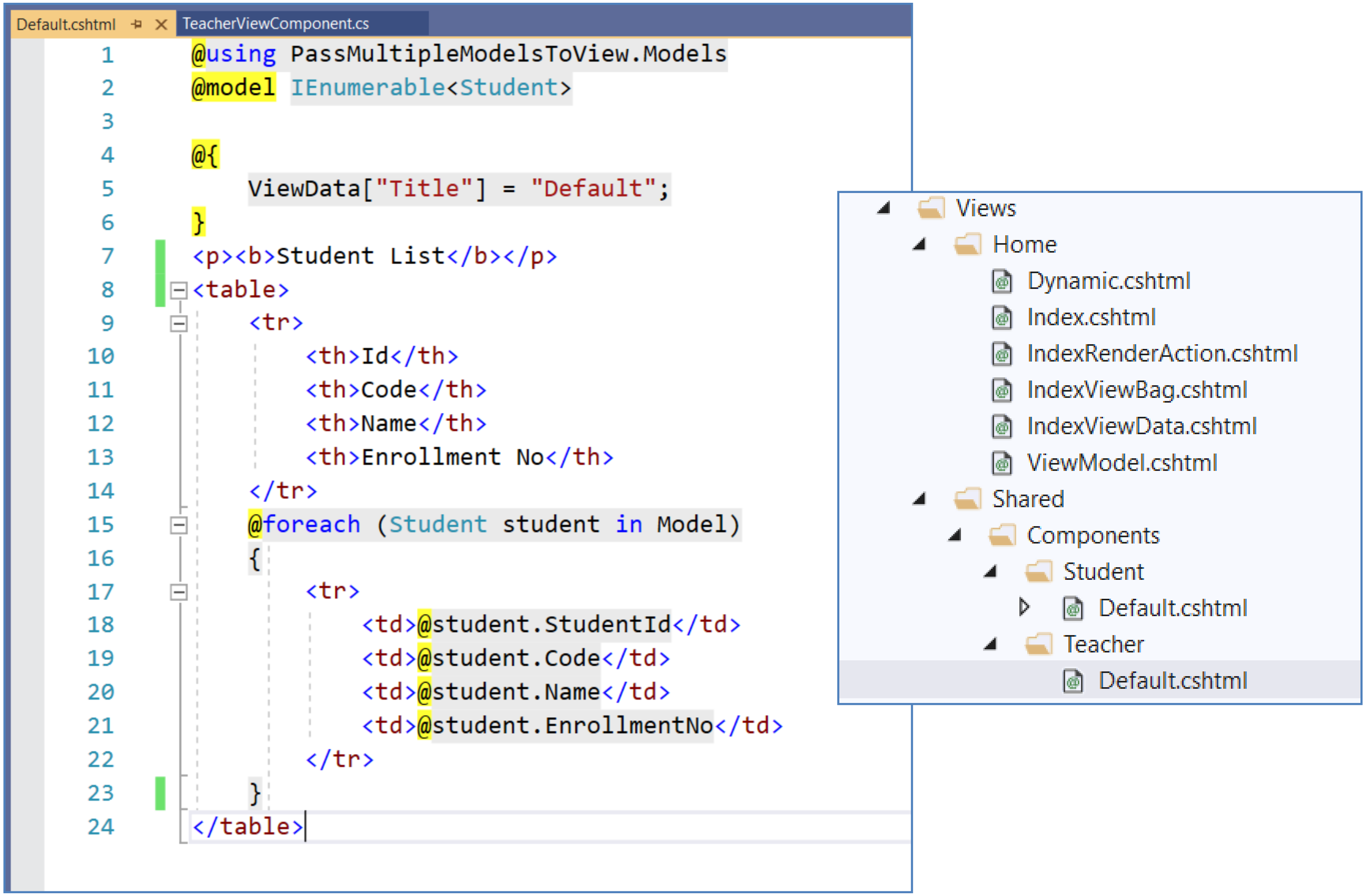
Template: List

Model class: Student (PassMultipleModelsToView.Models)

Options:

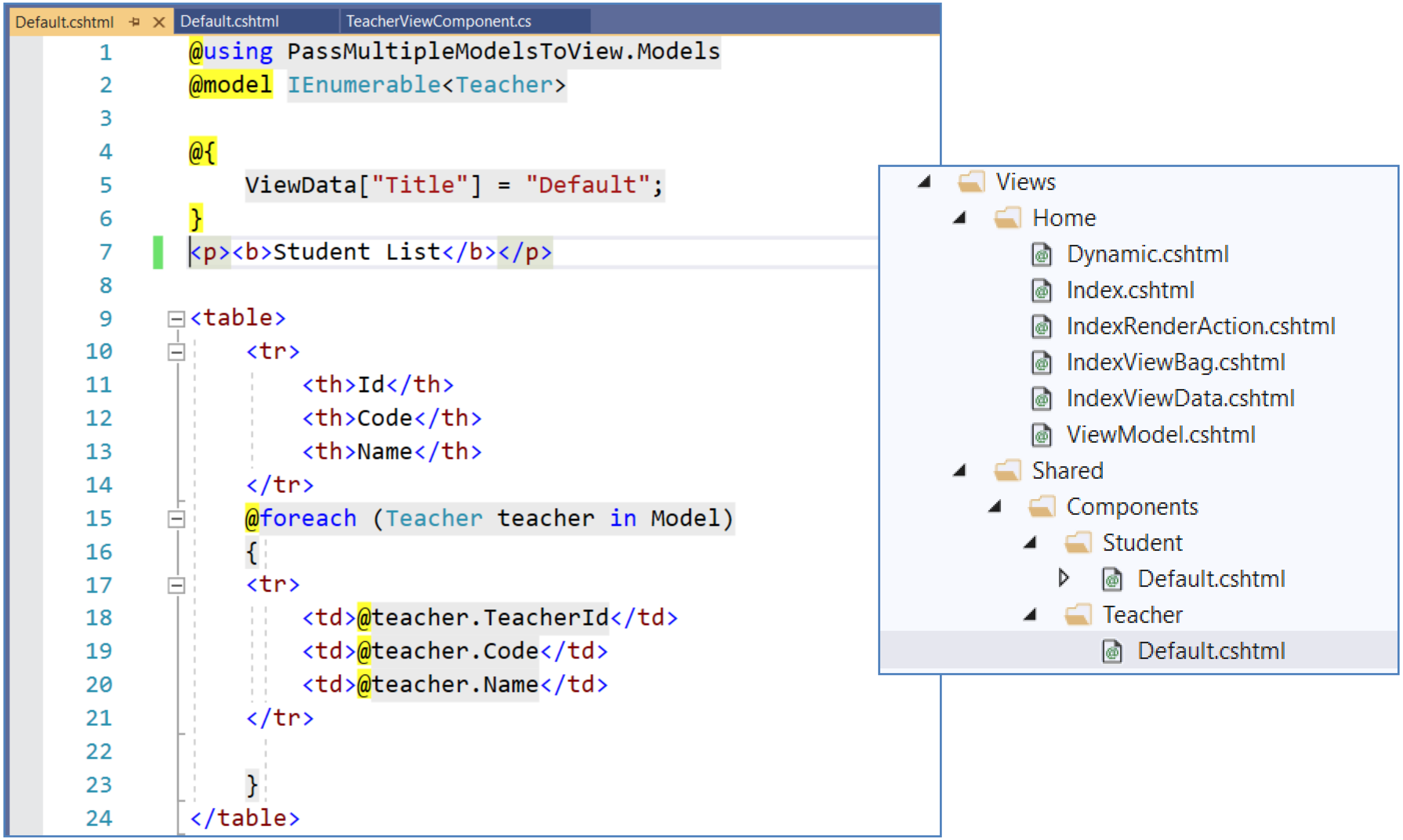
- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

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

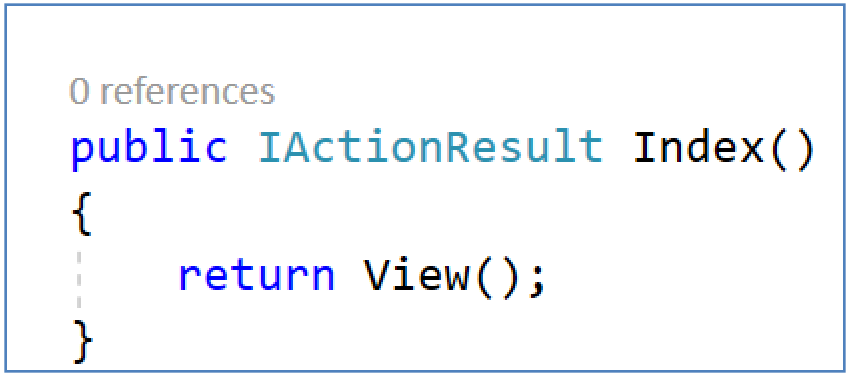


```
@using PassMultipleModelsToView.Models
@model IEnumerable<Student>

@{
    ViewData["Title"] = "Default";
}
<p><b>Student List</b></p>
<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
        <th>Enrollment No</th>
    </tr>
    @foreach (Student student in Model)
    {
        <tr>
            <td>@student.StudentId</td>
            <td>@student.Code</td>
            <td>@student.Name</td>
            <td>@student.EnrollmentNo</td>
        </tr>
    }
</table>
```

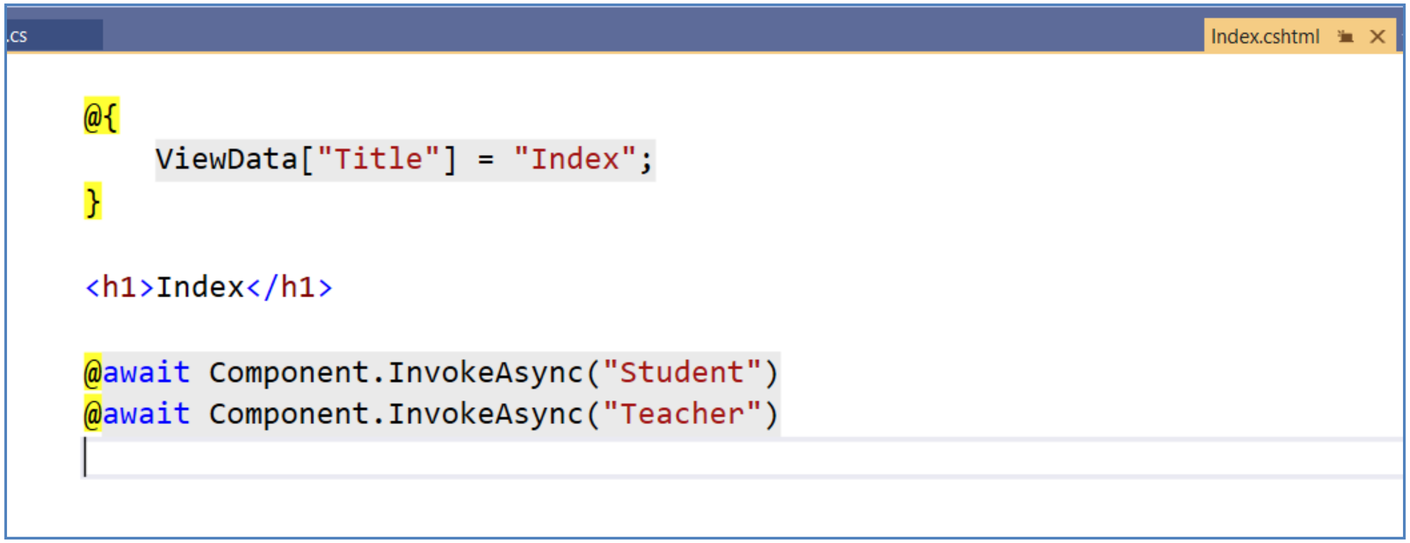
```
Views
    Home
        Dynamic.cshtml
        Index.cshtml
        IndexRenderAction.cshtml
        IndexViewBag.cshtml
        IndexViewData.cshtml
        ViewModel.cshtml
    Shared
        Components
            Student
                Default.cshtml
            Teacher
                Default.cshtml
```



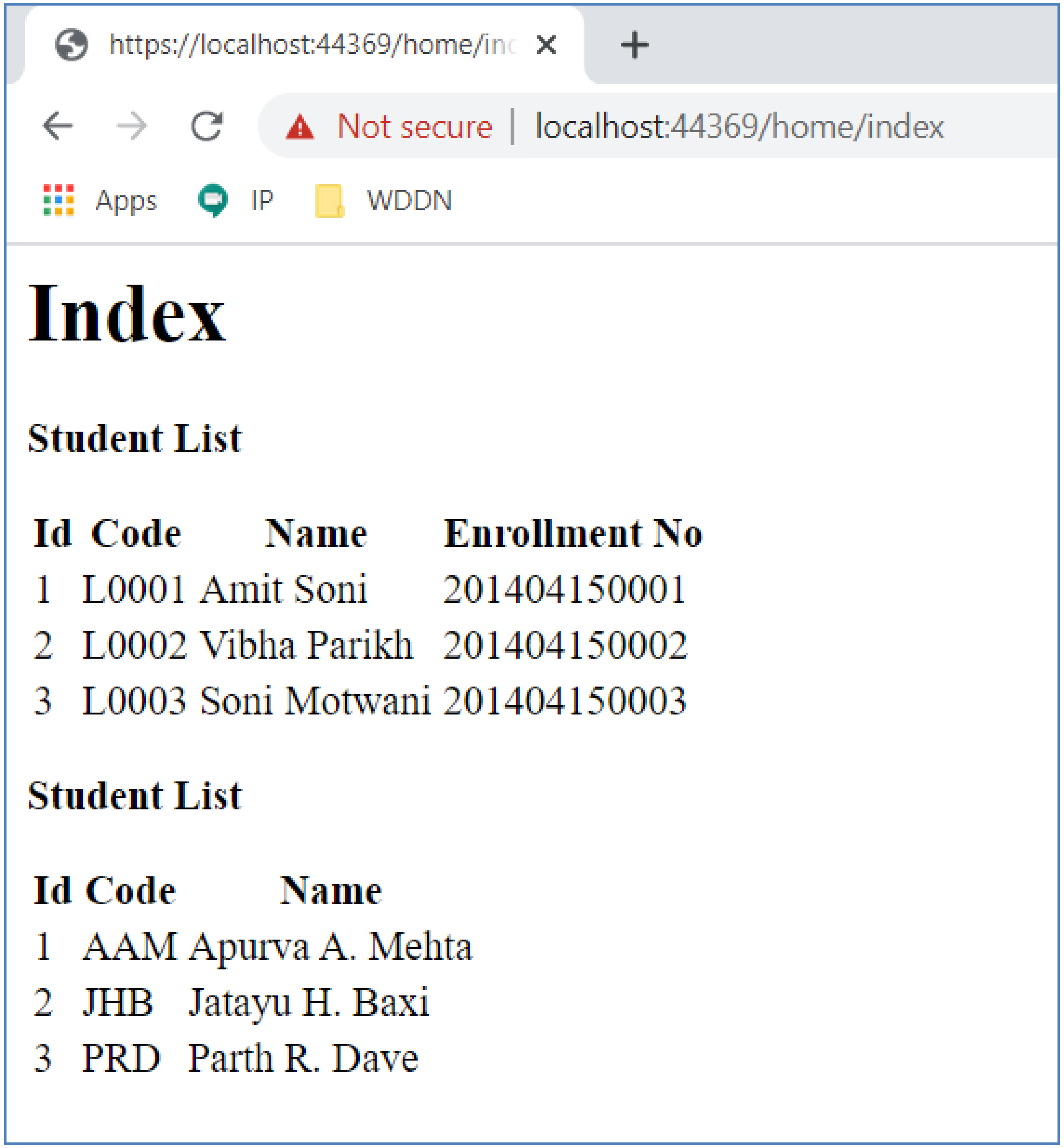
```
@using PassMultipleModelsToView.Models
@model IEnumerable<Teacher>

@{
    ViewData["Title"] = "Default";
}
<p><b>Student List</b></p>

<table>
    <tr>
        <th>Id</th>
        <th>Code</th>
        <th>Name</th>
    </tr>
    @foreach (Teacher teacher in Model)
    {
    <tr>
        <td>@teacher.TeacherId</td>
        <td>@teacher.Code</td>
        <td>@teacher.Name</td>
    </tr>

    }
</table>
```



```
0 references
public IActionResult Index()
{
    return View();
}
```



```
@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

@await Component.InvokeAsync("Student")
@await Component.InvokeAsync("Teacher")
```



```
Index

Student List

Id  Code    Name          Enrollment No
1   L0001   Amit Soni     201404150001
2   L0002   Vibha Parikh  201404150002
3   L0003   Soni Motwani  201404150003

Student List

Id  Code    Name
1   AAM     Apurva A. Mehta
2   JHB     Jatayu H. Baxi
3   PRD     Parth R. Dave
```

Popular use of View Component

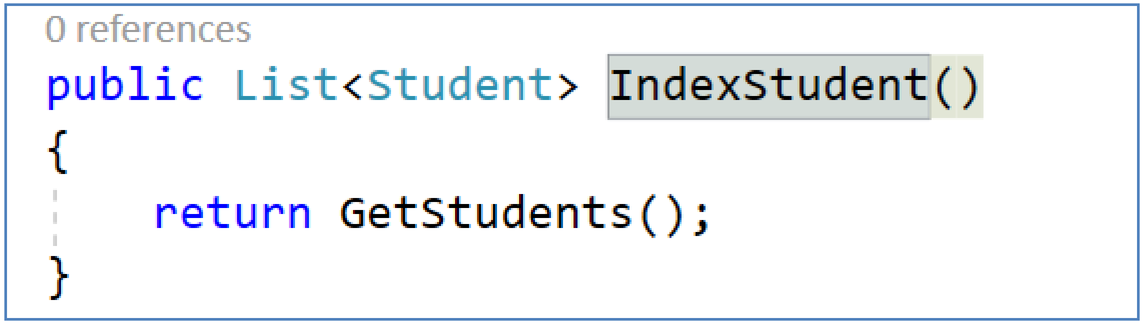
- View Components are intended anywhere you have reusable rendering logic that is too complex for a partial view, such as:
  - Dynamic navigation menus
  - Tag cloud (where it queries the database)
  - Login panel
  - Shopping cart
  - Recently published articles
  - Sidebar content on a typical blog
  - A login panel that would be rendered on every page and show either the links to log out or log in, depending on the log in state of the user

Controller action return types in ASP.NET Core

- ASP.NET Core offers the following options for controller action return types.
  - Specific type
  - IActionResult
  - ActionResult<T>

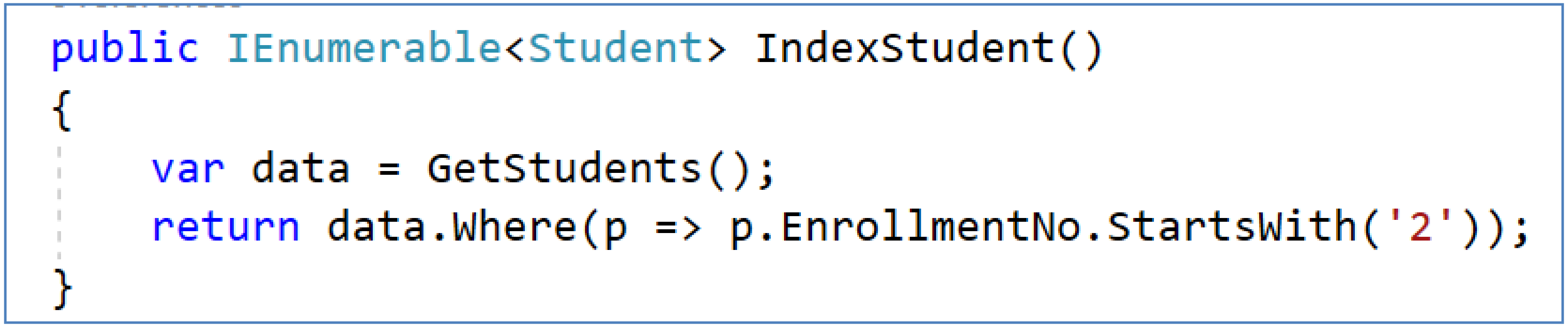
Specific type

- The simplest action returns a primitive or complex data type (for example, string or a custom object type).
- Without known conditions to safeguard against during action execution, returning a specific type could suffice.
  - The preceding action accepts no parameters, so parameter constraints validation isn't needed.



```
0 references
public List<Student> IndexStudent()
{
    return GetStudents();
}
```

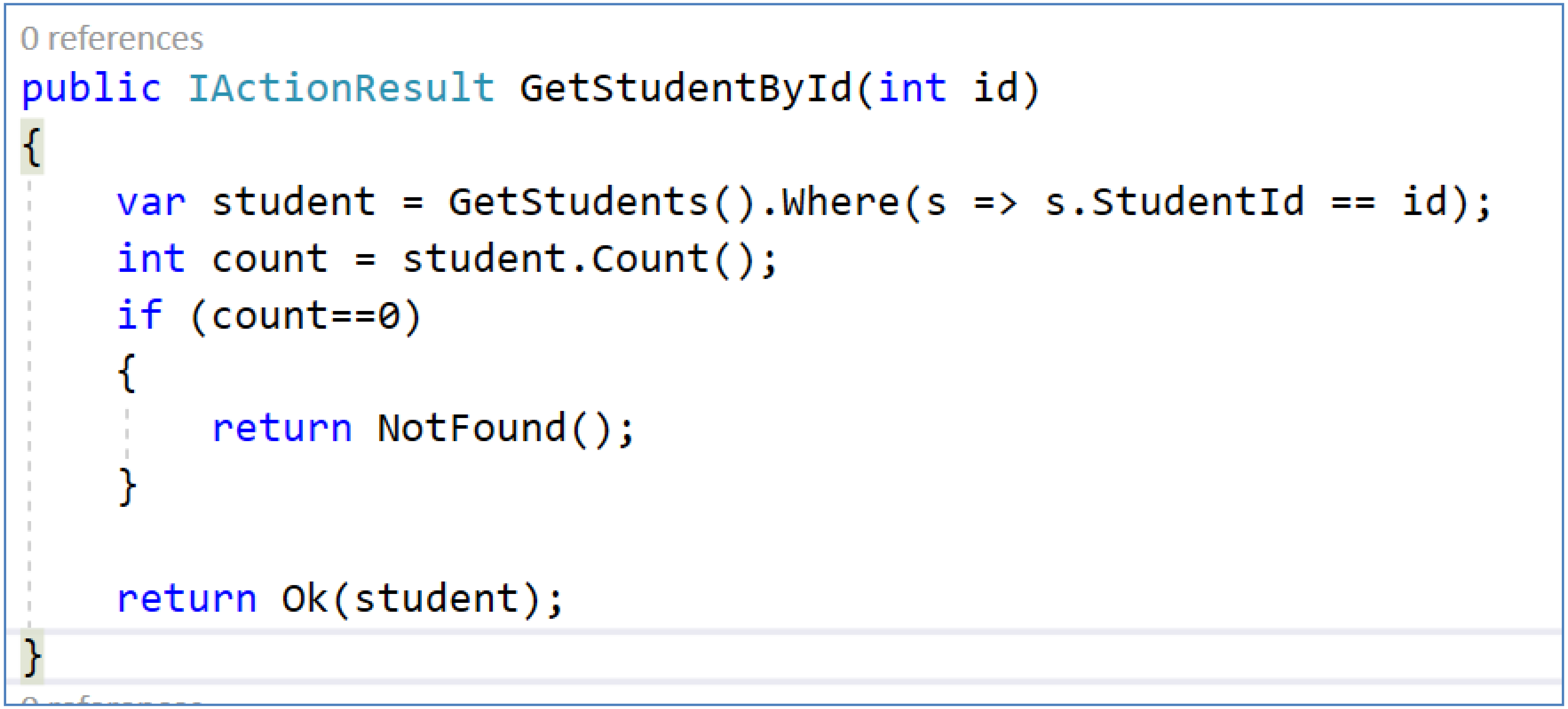
IEnumerable<T> or IAsyncEnumerable<T>



```
public IEnumerable<Student> IndexStudent()
{
    var data = GetStudents();
    return data.Where(p => p.EnrollmentNo.StartsWith('2'));
}
```

IActionResult type

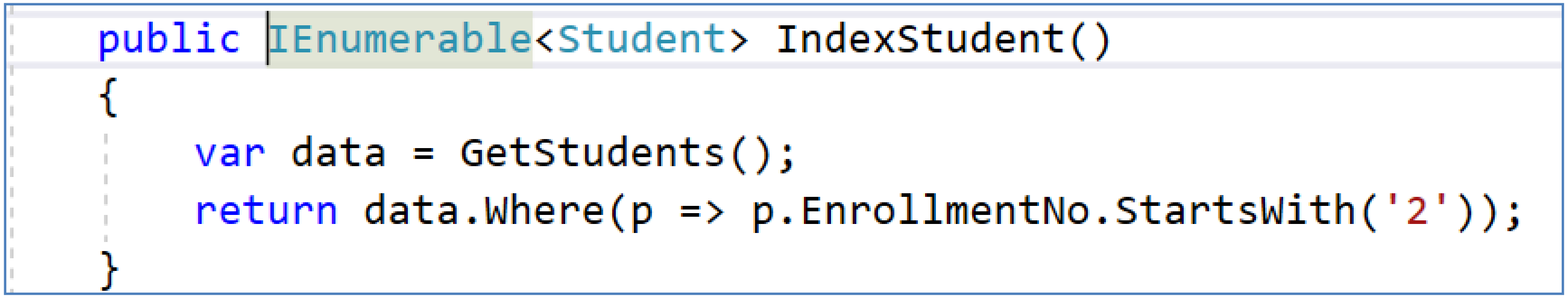
- The IActionResult return type is appropriate when multiple ActionResult return types are possible in an action.
- The ActionResult types represent various HTTP status codes.
- Any non-abstract class deriving from ActionResult qualifies as a valid return type.
  - Some common return types in this category are
    - BadRequestResult (400)
    - NotFoundResult (404)
    - OkObjectResult (200).
- Alternatively, convenience methods in the ControllerBase class can be used to return ActionResult types from an action.
  - For example, return BadRequest(); is a shorthand form of return new BadRequestResult();.



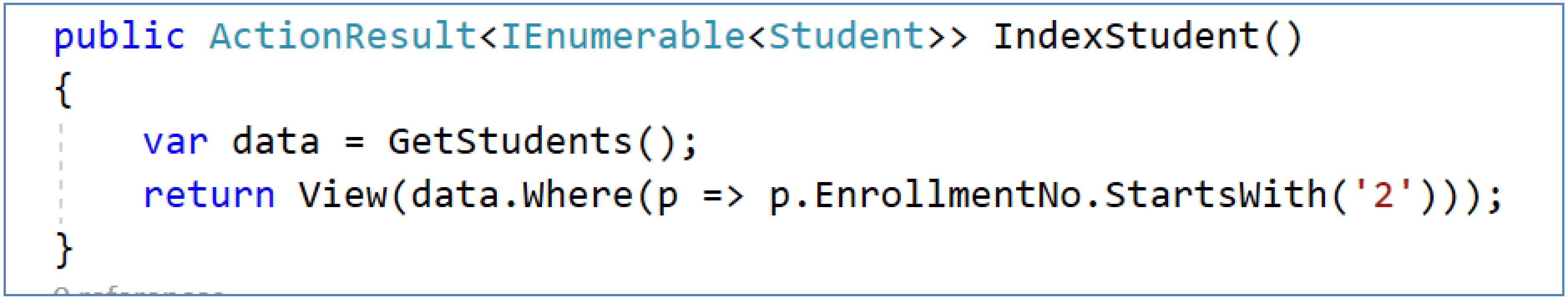
```
0 references
public IActionResult GetStudentById(int id)
{
    var student = GetStudents().Where(s => s.StudentId == id);
    int count = student.Count();
    if (count==0)
    {
        return NotFound();
    }

    return Ok(student);
}
```

ActionResult<T> type

- ASP.NET Core 2.1 introduced the ActionResult<T> return type for web controller actions.
- It enables you to return a type deriving from ActionResult or return a specific type.



```
public IEnumerable<Student> IndexStudent()
{
    var data = GetStudents();
    return data.Where(p => p.EnrollmentNo.StartsWith('2'));
}
```



```
public ActionResult<IEnumerable<Student>> IndexStudent()
{
    var data = GetStudents();
    return View(data.Where(p => p.EnrollmentNo.StartsWith('2')));
}
```

IActionResult Vs ActionResult

- IActionResult is an interface and ActionResult an implementation of that interface.
- IActionResult is an interface, and the platform is the one defining a type of a response
  - you can create a custom response, rather than just predefined ones for returning a View or a resource, here you can return a response, or error as well.
- ActionResult is an abstract class and action results like ViewResult, PartialViewResult, JsonResult, etc derive from ActionResult.

Filters in ASP.NET Core

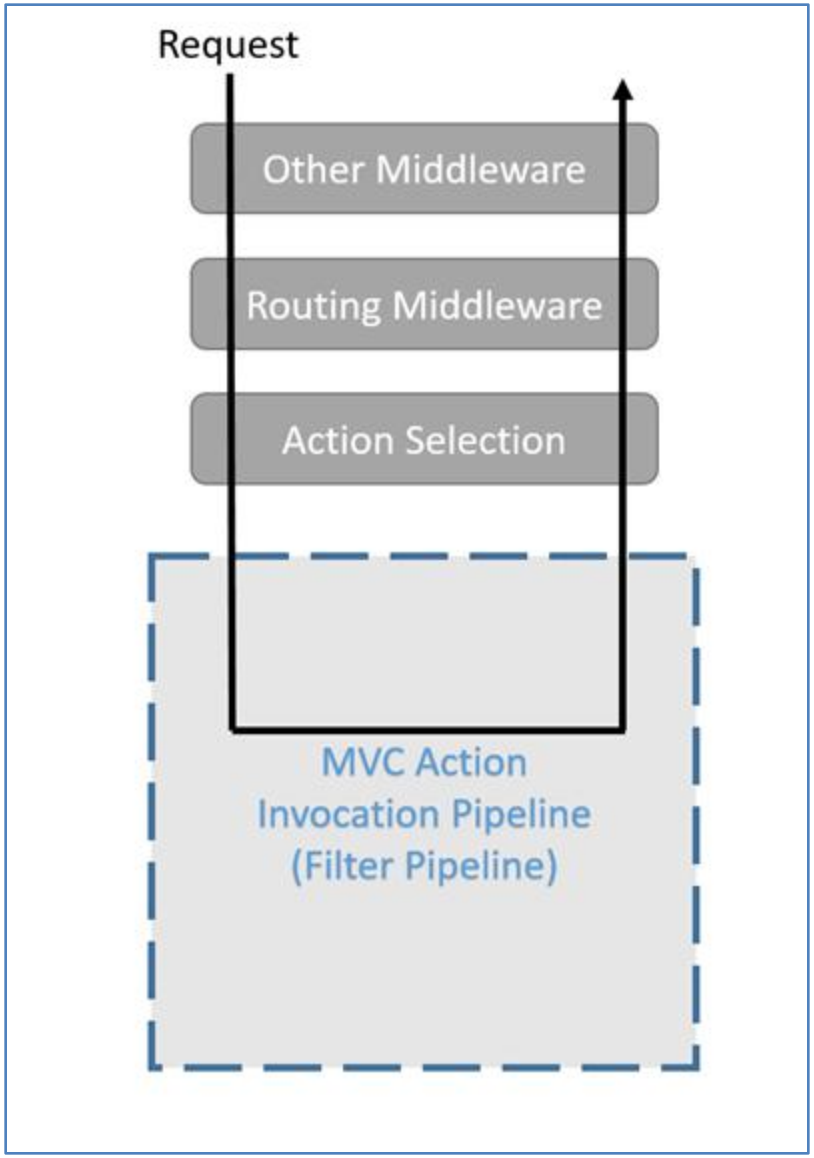
- *Filters* in ASP.NET Core allow code to be run before or after specific stages in the request processing pipeline.
- Built-in filters handle tasks such as:
  - Authorization (preventing access to resources a user isn't authorized for).
  - Response caching (short-circuiting the request pipeline to return a cached response).

Filters

- Custom filters can be created to handle cross-cutting concerns.
  - Examples of cross-cutting concerns include error handling, caching, configuration, authorization, and logging.
  - Filters avoid duplicating code. For example, an error handling exception filter could consolidate error handling.

How filters work

- Filters run within the *ASP.NET Core action invocation pipeline*, sometimes referred to as the *filter pipeline*.
- The filter pipeline runs after ASP.NET Core selects the action to execute.


Request
Other Middleware
Routing Middleware
Action Selection
MVC Action Invocation Pipeline (Filter Pipeline)

Filter types

- Authorization filters
- Resource filters
- Action filters
- Exception filters
- Result filters

Filter types

- Authorization filter run first and are used to determine whether the user is authorized for the request. Authorization filters short-circuit the pipeline if the request is not authorized.
- Resource filters
- Action filters
- Exception filters
- Result filters

Filter types

- Authorization filters
- Resource filters
  - Run after authorization.
  - OnResourceExecuting runs code before the rest of the filter pipeline. For example,

    OnResourceExecuting runs code before model binding.
  - OnResourceExecuted runs code after the rest of the pipeline has completed.
- Action filters
- Exception filters
- Result filters

Filter types

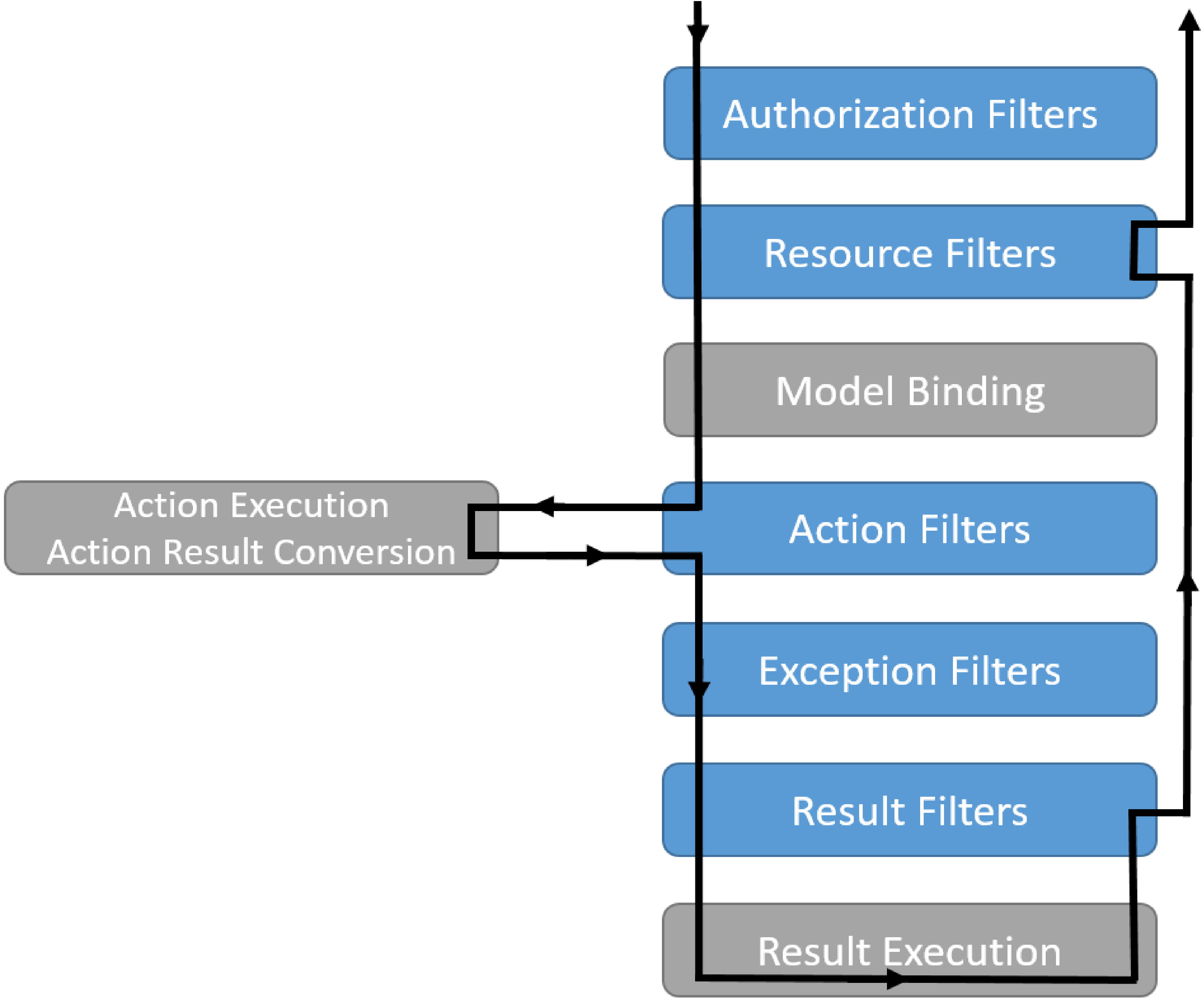
- Authorization filters
- Resource filters
- Action filters
  - Run code immediately before and after an action method is called.
  - Can change the arguments passed into an action.
  - Can change the result returned from the action.
  - Are **not** supported in Razor Pages.
- Exception filters
- Result filters

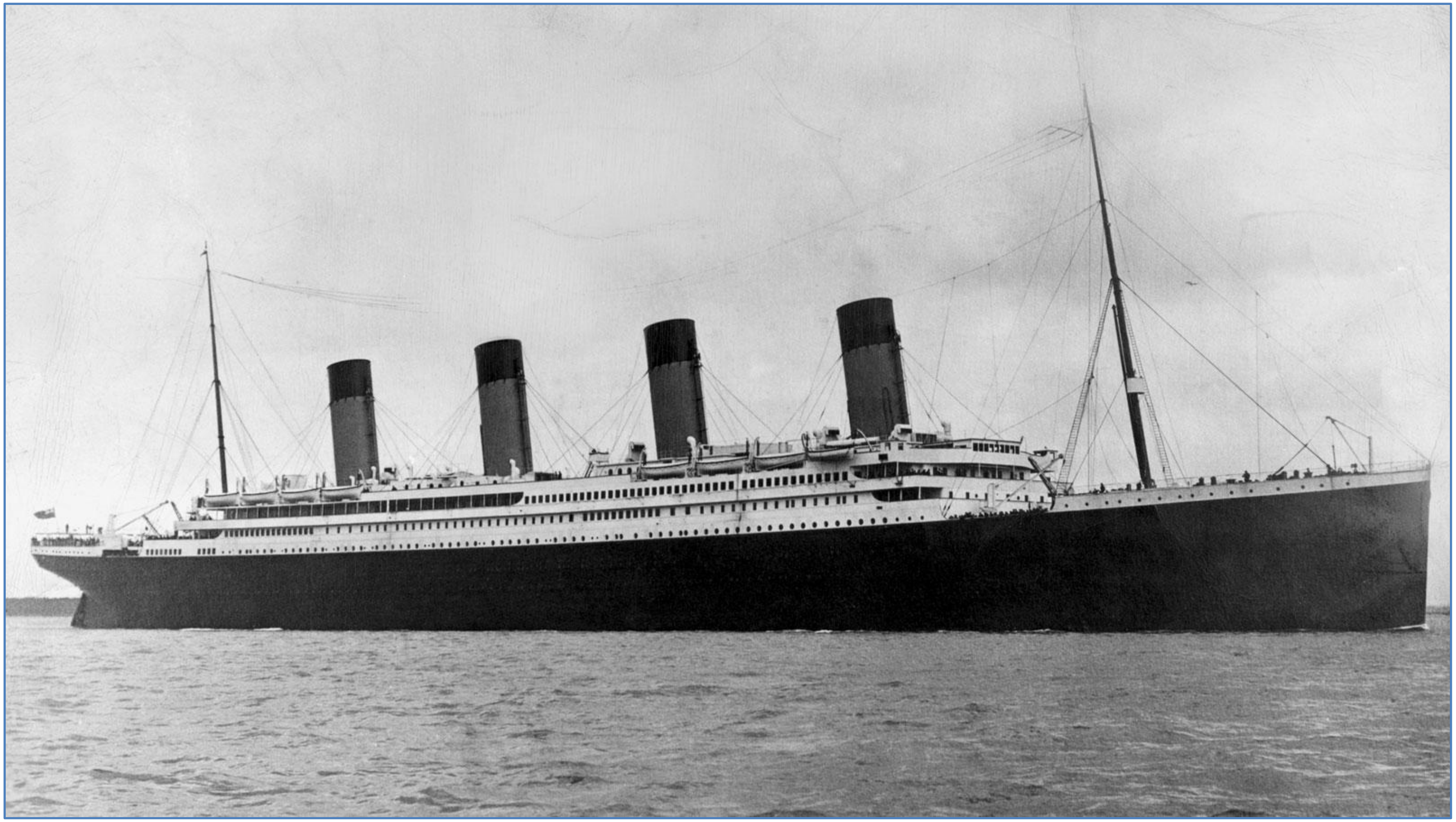
Filter types

- Authorization filters
- Resource filters
- Action filters
- Exception filters apply global policies to unhandled exceptions that occur before the response body has been written to.
- Result filters

Filter types

- Authorization filters
- Resource filters
- Action filters
- Exception filters
- Result filters run code immediately before and after the execution of action results. They run only when the action method has executed successfully. They are useful for logic that must surround view or formatter execution.

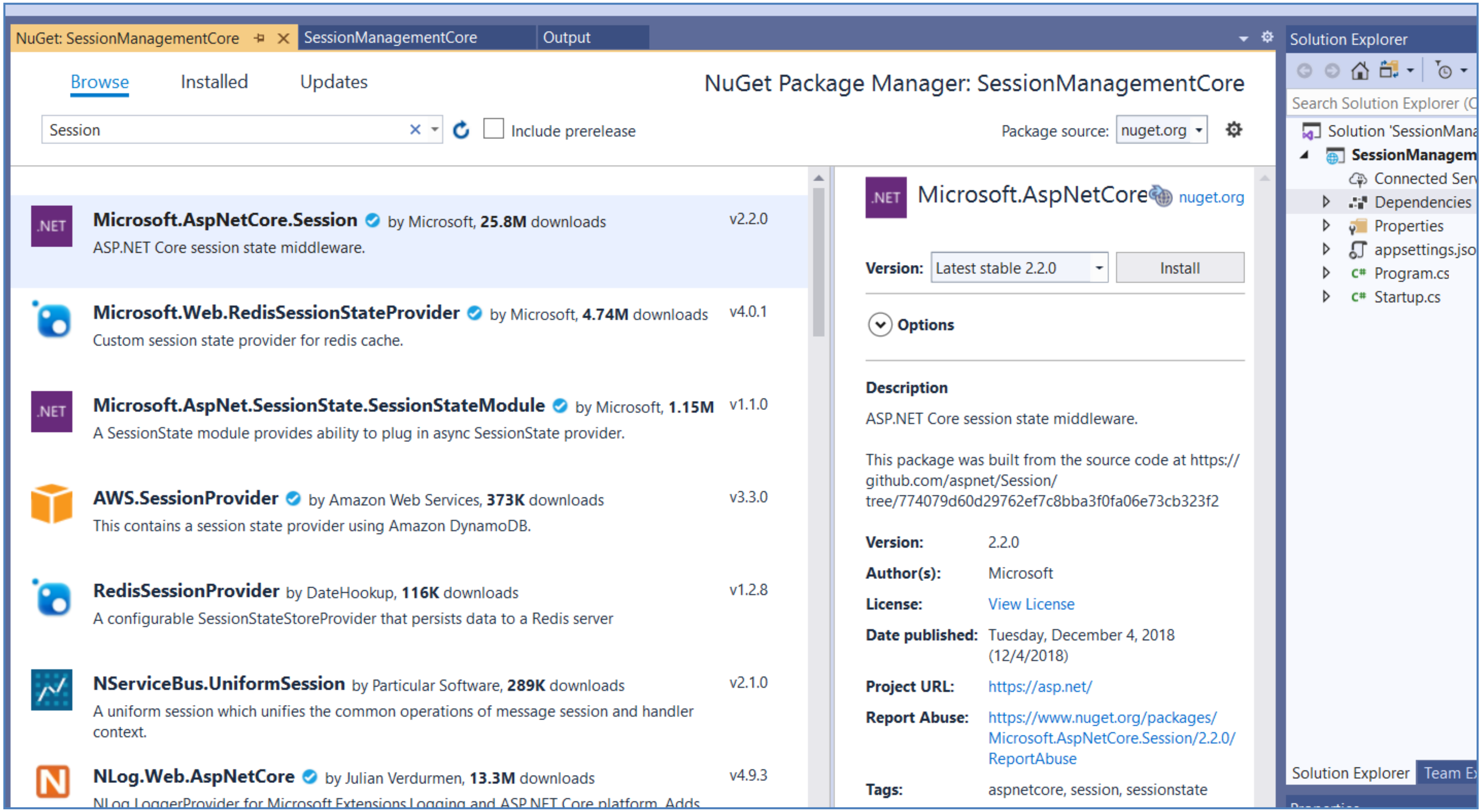

Authorization Filters
Resource Filters
Model Binding
Action Execution
Action Result Conversion
Action Filters
Exception Filters
Result Filters
Result Execution



Session and state management in ASP.NET Core

- HTTP is a stateless protocol.
- By default, HTTP requests are independent messages that don't retain user values.

Storage approach	Storage mechanism
Cookies	HTTP cookies. May include data stored using server-side app code.
Session state	HTTP cookies and server-side app code
TempData	HTTP cookies or session state
Query strings	HTTP query strings
Hidden fields	HTTP form fields
HttpContext.Items	Server-side app code
Cache	Server-side app code


NuGet: SessionManagementCore | SessionManagementCore | Output

Browse | Installed | Updates

NuGet Package Manager: SessionManagementCore

Session | Include prerelease | Package source: nuget.org

Microsoft.AspNetCore.Session by Microsoft, **25.8M** downloads v2.2.0
ASP.NET Core session state middleware.

Microsoft.Web.RedisSessionStateProvider by Microsoft, **4.74M** downloads v4.0.1
Custom session state provider for redis cache.

Microsoft.AspNet.SessionState.SessionStateModule by Microsoft, **1.15M** v1.1.0
A SessionState module provides ability to plug in async SessionState provider.

AWS.SessionProvider by Amazon Web Services, **373K** downloads v3.3.0
This contains a session state provider using Amazon DynamoDB.

RedisSessionProvider by DateHookup, **116K** downloads v1.2.8
A configurable SessionStateStoreProvider that persists data to a Redis server

NServiceBus.UniformSession by Particular Software, **289K** downloads v2.1.0
A uniform session which unifies the common operations of message session and handler context.

NLog.Web.AspNetCore by Julian Verdurmen, **13.3M** downloads v4.9.3
NLog LoggerProvider for Microsoft.Extensions.Logging and ASP.NET Core platform. Adds

Microsoft.AspNetCore nuget.org

Version: Latest stable 2.2.0 | Install

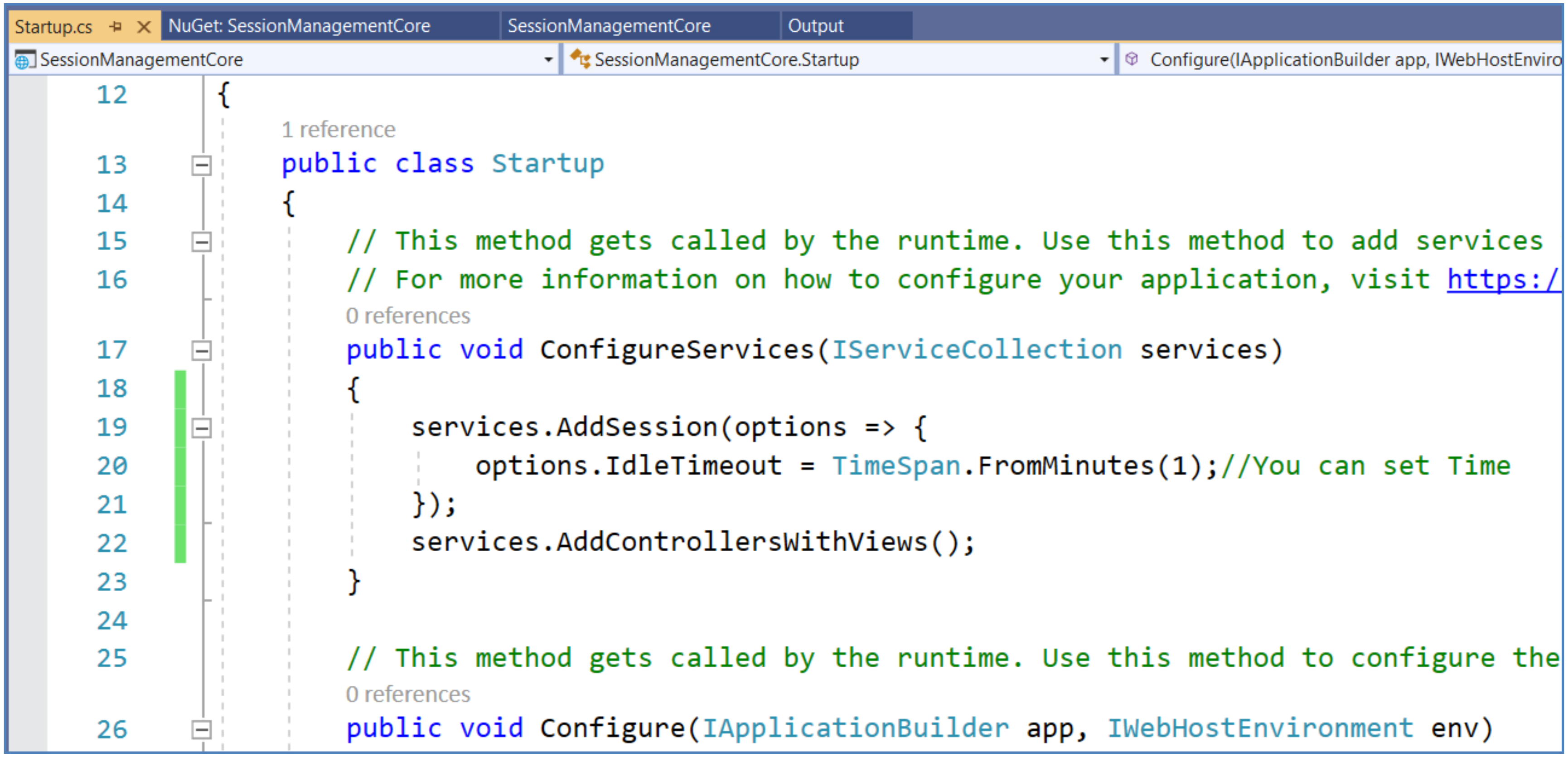
Options

Description

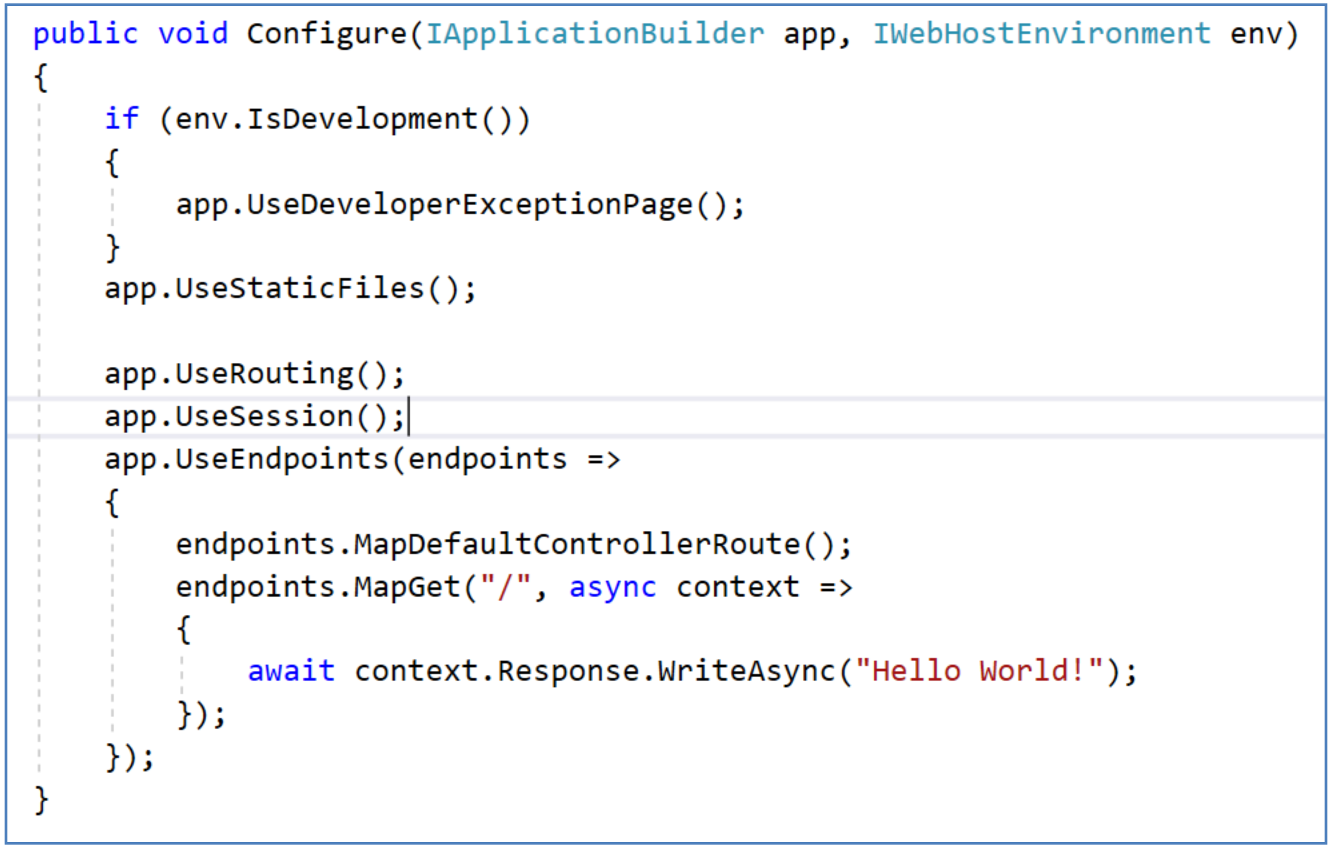
ASP.NET Core session state middleware.

This package was built from the source code at https://github.com/aspnet/Session/tree/774079d60d29762ef7c8bba3f0fa06e73cb323f2

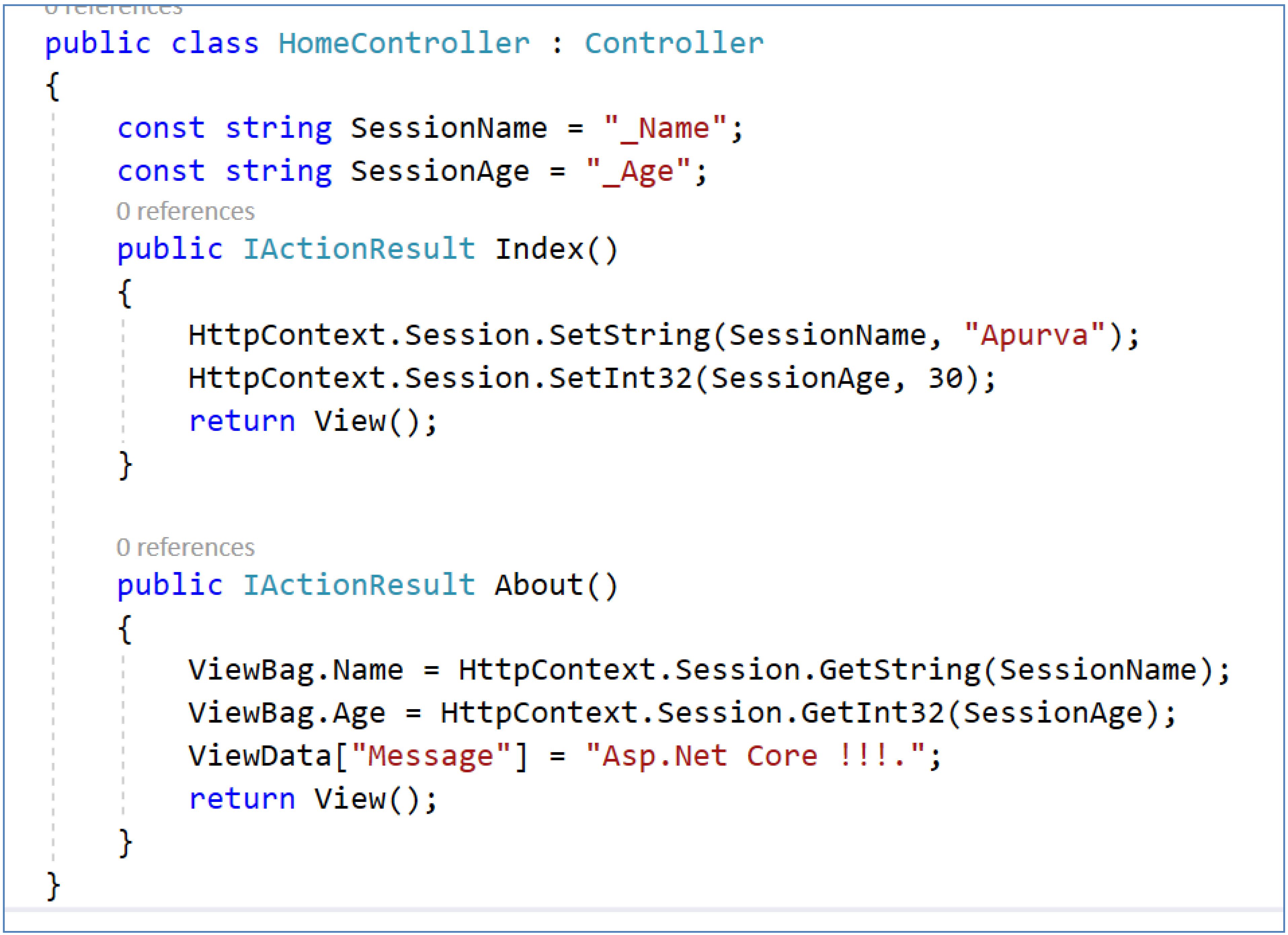
Version:	2.2.0
Author(s):	Microsoft
License:	View License
Date published:	Tuesday, December 4, 2018 (12/4/2018)
Project URL:	https://asp.net/
Report Abuse:	https://www.nuget.org/packages/Microsoft.AspNetCore.Session/2.2.0/ReportAbuse
Tags:	aspnetcore, session, sessionstate


Startup.cs
NuGet: SessionManagementCore
SessionManagementCore
Output
SessionManagementCore
SessionManagementCore.Startup
Configure(IApplicationBuilder app, IWebHostEnviro

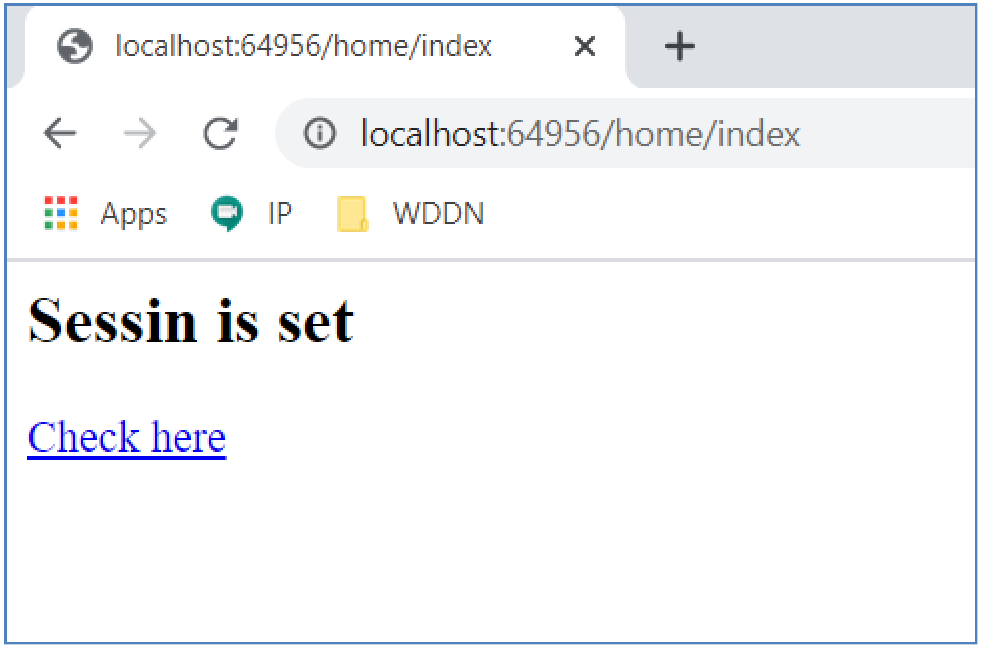
```
{
    1 reference
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services
        // For more information on how to configure your application, visit https:/
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSession(options => {
                options.IdleTimeout = TimeSpan.FromMinutes(1);//You can set Time
            });
            services.AddControllersWithViews();
        }

        // This method gets called by the runtime. Use this method to configure the
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```



```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.UseStaticFiles();

    app.UseRouting();
    app.UseSession();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    });
}
```



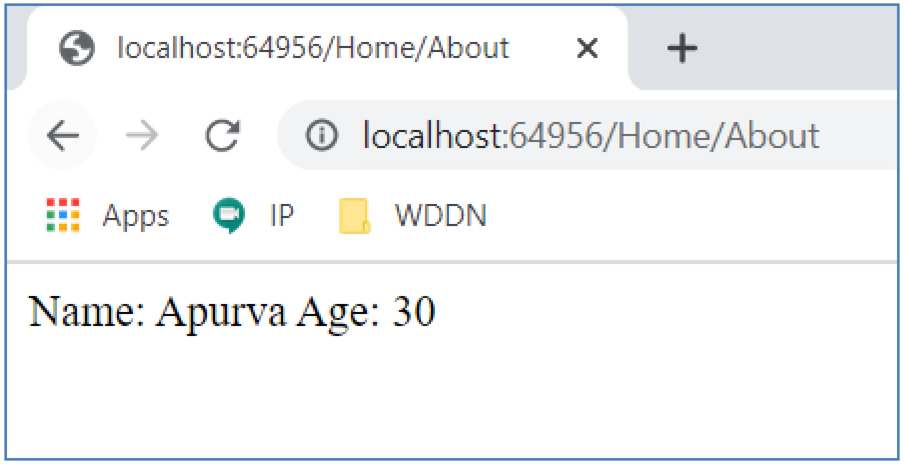
```
0 references
public class HomeController : Controller
{
    const string SessionName = "_Name";
    const string SessionAge = "_Age";
    0 references
    public IActionResult Index()
    {
        HttpContext.Session.SetString(SessionName, "Apurva");
        HttpContext.Session.SetInt32(SessionAge, 30);
        return View();
    }

    0 references
    public IActionResult About()
    {
        ViewBag.Name = HttpContext.Session.GetString(SessionName);
        ViewBag.Age = HttpContext.Session.GetInt32(SessionAge);
        ViewData["Message"] = "Asp.Net Core !!!.";
        return View();
    }
}
```


localhost:64956/home/index

Apps IP WDDN

Sessin is set

Check here


localhost:64956/Home/About

Apps IP WDDN

Name: Apurva Age: 30

Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core

- Cross-site request forgery (XSRF or CSRF) is an attack against web-hosted apps whereby a malicious web app can influence the interaction between a client browser and a web app that trusts that browser.

- These attacks are possible because web browsers send some types of authentication tokens automatically with every request to a website.

- one-click attack or session riding

CSRF Example

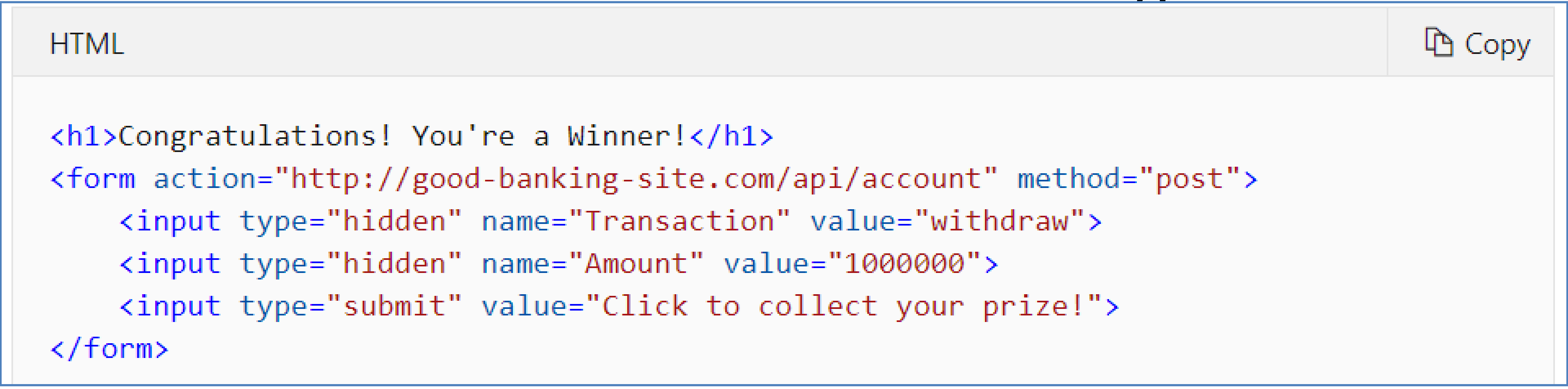
1. A user signs into

   www.good-banking-site.com using forms authentication. The server authenticates the user and issues a response that includes an authentication cookie. The site is vulnerable to attack because it trusts any request that it receives with a valid authentication cookie.
2. The user visits a malicious site,

   www.bad-crook-site.com.

CSRF Example

2. The user visits a malicious site, www.bad-crook-site.com. The malicious site, www.bad-crook-site.com, contains an HTML form similar to the following:



```html
<h1>Congratulations! You're a Winner!</h1>
<form action="http://good-banking-site.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw">
    <input type="hidden" name="Amount" value="1000000">
    <input type="submit" value="Click to collect your prize!">
</form>
```

Notice that the form's action posts to the vulnerable site, not to the malicious site. This is the "cross-site" part of CSRF.

CSRF Example

3. The user selects the submit button. The browser makes the request and automatically includes the authentication cookie for the requested domain,

   www.good-banking-site.com.

3. The request runs on the www.good-banking-site.com server with the user's authentication context and can perform any action that an authenticated user is allowed to perform.

CSRF also possible with…

- In addition to the scenario where the user selects the button to submit the form, the malicious site could:
  - Run a script that automatically submits the form.
  - Send the form submission as an AJAX request.
  - Hide the form using CSS.
- These alternative scenarios don't require any action or input from the user other than initially visiting the malicious site.
- Using HTTPS doesn't prevent a CSRF attack. The malicious site can send

  an https://www.good-banking-site.com/ request just as easily as it can send an insecure request.

Cookies not the safe…

- CSRF attacks are possible against web apps that use cookies for authentication because:
  - Browsers store cookies issued by a web app.
  - Stored cookies include session cookies for authenticated users.
  - Browsers send all of the cookies associated with a domain to the web app every request regardless of how the request to app was generated within the browser.

What a user can do?

- Users can guard against CSRF vulnerabilities by taking precautions:
  - Sign off of web apps when finished using them.
  - Clear browser cookies periodically.
- However, CSRF vulnerabilities are fundamentally a problem with the web app, not the end user!!!

Authentication fundamentals

- Cookie-based authentication
- Token-based authentication

Cookie-based authentication

- When a user authenticates using their username and password, they're issued a token, containing an authentication ticket that can be used for authentication and authorization.
- The token is stored as a cookie that accompanies every request the client makes.
- Generating and validating this cookie is performed by the Cookie Authentication Middleware.

Cookie-based authentication

- The middleware serializes a user principal into an encrypted cookie.
- On subsequent requests, the middleware validates the cookie, recreates the principal, and assigns the principal to the User property of HttpContext.

Token-based authentication

- When a user is authenticated, they're issued a token (not an antiforgery token).
- The token contains user information in the form of claims or a reference token that points the app to user state maintained in the app.
- When a user attempts to access a resource requiring authentication, the token is sent to the app with an additional authorization header in form of Bearer token.
- This makes the app stateless.

Token-based authentication

- In each subsequent request, the token is passed in the request for server-side validation.
- This token isn't *encrypted*; it's *encoded*.
- On the server, the token is decoded to access its information.
- To send the token on subsequent requests, store the token in the browser's local storage.
- Don't be concerned about CSRF vulnerability if the token is stored in the browser's local storage.
- CSRF is a concern when the token is stored in a cookie.

Multiple apps hosted at one domain are vulnerable

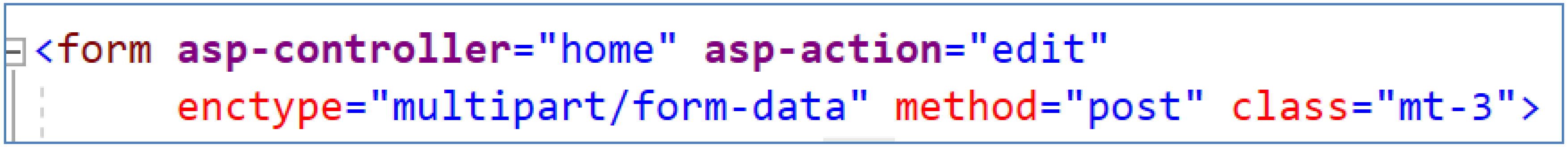
- Shared hosting environments are vulnerable to session hijacking, login CSRF, and other attacks.
- Although example1.contoso.net and example2.contoso .net are different hosts, there's an implicit trust relationship between hosts under the *.contoso.net domain.
- This implicit trust relationship allows potentially untrusted hosts to affect each other‘s.
- Attacks that exploit trusted cookies between apps hosted on the same domain can be prevented by not sharing domains.
- When each app is hosted on its own domain, there is no implicit cookie trust relationship to exploit.

ASP.NET Core antiforgery configuration

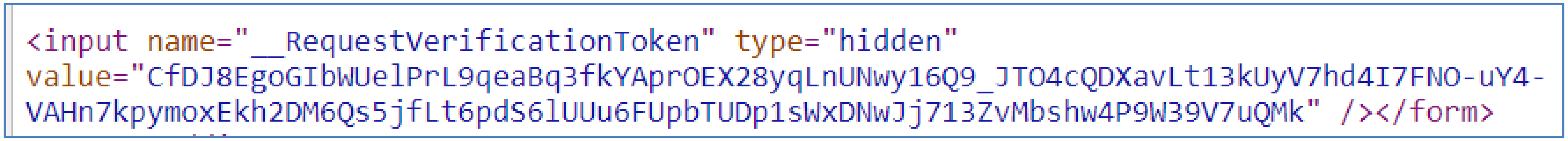
- Antiforgery middleware is added to the Dependency injection container when one of the following APIs is called in Startup.ConfigureServices
  - AddMvc
  - MapRazorPages
  - MapControllerRoute
  - MapBlazorHub
  - AddControllersWithViews

Cont.

- In ASP.NET Core 2.0 or later, the FormTagHelper injects antiforgery tokens into HTML form elements.
- The following markup in a Razor file automatically generates antiforgery tokens:



```
<form asp-controller="home" asp-action="edit"
      enctype="multipart/form-data" method="post" class="mt-3">
```



```
<input name="__RequestVerificationToken" type="hidden"
value="CfDJ8EgoGIbWUelPrL9qeaBq3fkYAprOEX28yqLnUNwy16Q9_JTO4cQDXavLt13kUyV7hd4I7FNO-uY4-
VAHn7kpymoxEkh2DM6Qs5jfLt6pdS6lUUu6FUpbTUDp1sWxDNwJj713ZvMbshw4P9W39V7uQMk" /></form>
```

Synchronizer Token Pattern (STP)

- The most common approach to defending against CSRF attacks is to use the *Synchronizer Token Pattern* (STP).
- STP is used when the user requests a page with form data:

1. The server sends a token associated with the current user's identity to the client.
2. The client sends back the token to the server for verification.
3. If the server receives a token that doesn't match the authenticated user's identity, the request is rejected.

Synchronizer Token Pattern (STP)

- The token is unique and unpredictable.
- The token can also be used to ensure proper sequencing of a series of requests (for example, ensuring the request sequence of: page 1 > page 2 > page 3).
- All of the forms in ASP.NET Core MVC and Razor Pages templates generate antiforgery tokens.

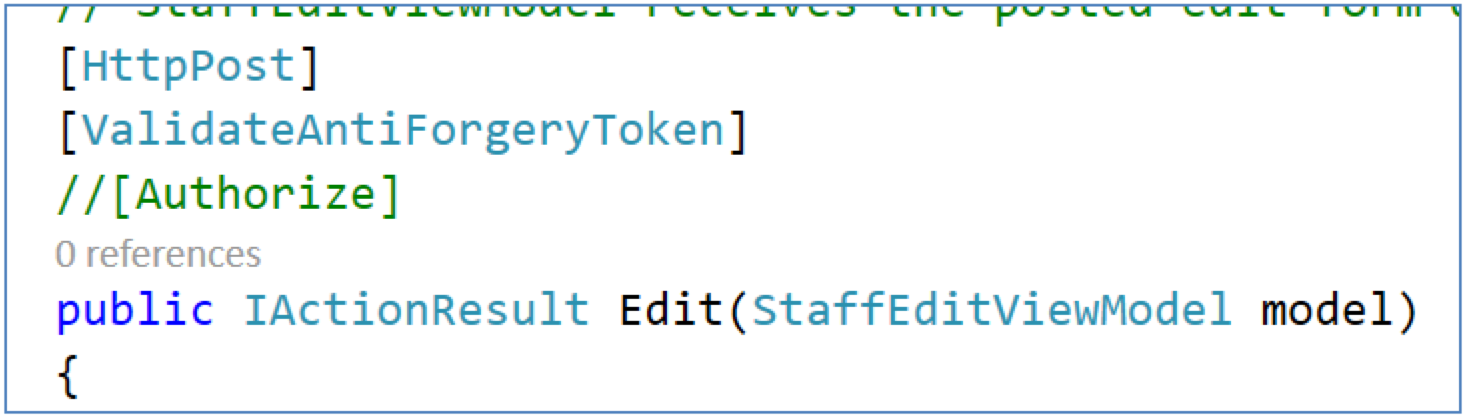
Filters for working with antiforgery tokens

- ASP.NET Core includes three filters for working with antiforgery tokens:

1. ValidateAntiForgeryToken
2. AutoValidateAntiforgeryToken
3. IgnoreAntiforgeryToken

Require antiforgery validation

- ValidateAntiForgeryToken is an action filter that can be applied to an individual action, a controller, or globally.
- Requests made to actions that have this filter applied are blocked unless the request includes a valid antiforgery token.



```
// StaffEditViewModel receives the posted edit form
[HttpPost]
[ValidateAntiForgeryToken]
//[Authorize]
0 references
public IActionResult Edit(StaffEditViewModel model)
{
```

- The ValidateAntiForgeryToken attribute requires a token for requests to the action methods it marks, including HTTP GET requests.
- If the ValidateAntiForgeryToken attribute is applied across the app's controllers, it can be overridden with the IgnoreAntiforgeryToken attribute.

Automatically validate antiforgery tokens for unsafe HTTP methods only

- ASP.NET Core apps don't generate antiforgery tokens for safe HTTP methods (GET, HEAD, OPTIONS, and TRACE).
- Instead of broadly applying the **ValidateAntiForgeryToken** attribute and then overriding it with **IgnoreAntiforgeryToken** attributes, the **AutoValidateAntiforgeryToken** attribute can be used.
- **AutoValidateAntiforgeryToken** attribute works identically to the ValidateAntiForgeryToken attribute, except that it doesn't require tokens for requests made using the HTTP methods like GET, HEAD, OPTIONS , and TRACE.

Cont.

- It is recommend use of AutoValidateAntiforgeryToken broadly for non-API scenarios.
- This ensures POST actions are protected by default.
- The alternative is to ignore antiforgery tokens by default, unless ValidateAntiForgeryToken is applied to individual action methods.
- It's more likely in this scenario for a POST action method to be left unprotected by mistake, leaving the app vulnerable to CSRF attacks. All POSTs should send the antiforgery token.

Publish an ASP.NET Core app to IIS

- https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/publish-to-iis?view=aspnetcore-3.1&tabs=visual-studio