ASP.NET Web API

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

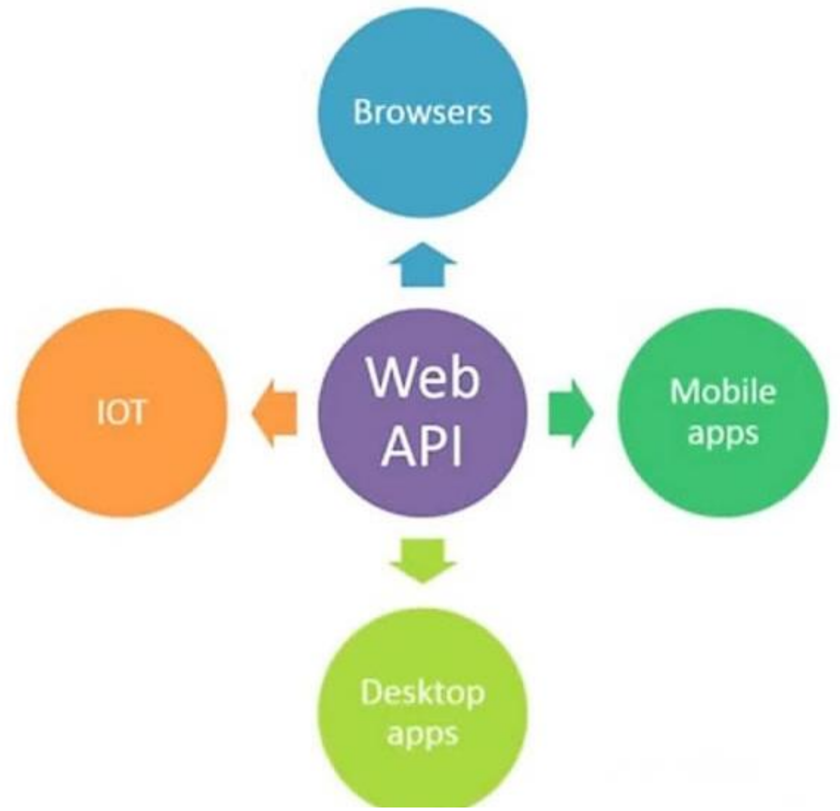
Agenda

- ASP.NET Web API
  - Overview of ASP.NET Web API
  - API Controllers
  - Writing Action Methods in Web Api
  - Attribute Routing
  - HTTP Verb [HttpGet], [HttpPost], [HttpPut], and[HttpDelete]
  - Web API Result Types
  - Parameter Binding
  - What is Cors
  - How to Implement Cors
  - Error Handling
- Security
  - Authentication
    - Windows
    - Forms
  - Authorization
  - Cross-Site Scripting (XSS)
  - Cross site request forgery (CSRF)
- Unit Testing
  - Testing an ASP.NET MVC Controller using NUnit
  - Creating an ASP.NET MVC Application
  - Creating Test Projects
  - Installing NUnit Framework and adding in the project
  - Testing the Controller

ASP.NET Web API

Overview of ASP.NET Web API

- API (Application Programing Interface).
- In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building software and applications.
- Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc. For example, Twitter's REST APIs provide programmatic access to read and write data using which we can integrate twitter's capabilities into our own application.



Overview of ASP.NET Web API

ASP.NET Web API

The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view. It is like a webservice or WCF service but the exception is that it only supports HTTP protocol.

ASP.NET Web API Characteristics

- ASP.NET Web API is an ideal platform for building RESTful services.
- ASP.NET Web API is built on top of ASP.NET and supports ASP.NET request/response pipeline
- ASP.NET Web API maps HTTP verbs to method names.
- ASP.NET Web API supports different formats of response data. Built-in support for JSON, XML, BSON format.
- ASP.NET Web API can be hosted in IIS, Self-hosted or other web server that supports .NET 4.0+.
- ASP.NET Web API framework includes new HttpClient to communicate with Web API server. HttpClient can be used in ASP.MVC server side, Windows Form application, Console application or other apps.

Overview of ASP.NET Web API

When to choose ASP.NET Web API?

- Choose Web API if you are using .NET framework 4.0 or above.
- Choose Web API if you want to build a service that supports only HTTP protocol.
- Choose Web API to build RESTful HTTP based services.
- Choose Web API if you are familiar with ASP.NET MVC.

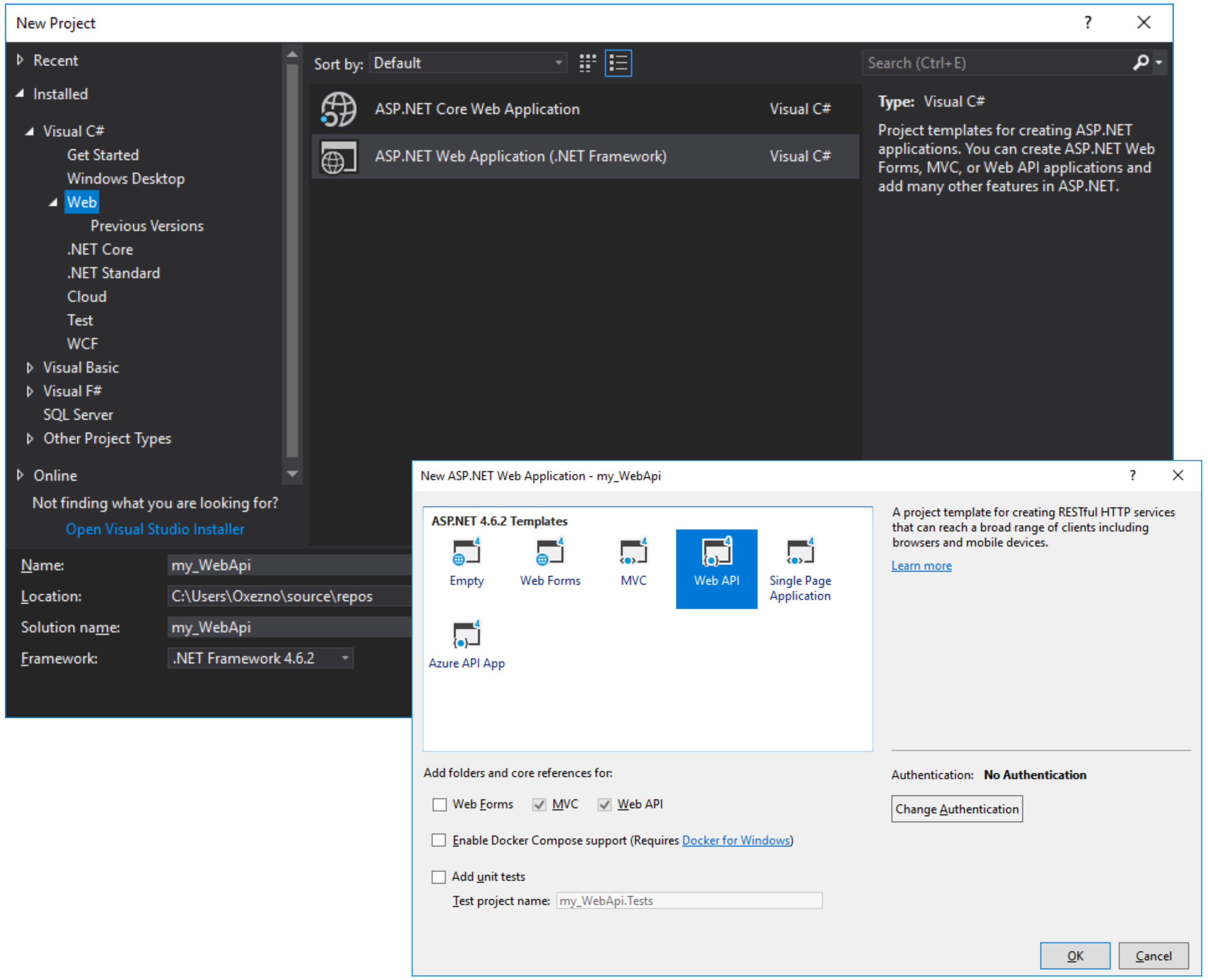
Create Web API project

Create a new Application

Start your Visual Studio and select:

File → New → Project. Select Web → ASP.NET Web Application (.NET Framework) → Web API

Name this project as **my_WebApi**



New Project

Sort by: Default

Search (Ctrl+E)

Not finding what you are looking for?

Open Visual Studio Installer

ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#

Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: my_WebApi

Location: C:\Users\Oxezno\source\repos

Solution name: my_WebApi

Framework: .NET Framework 4.6.2

New ASP.NET Web Application - my_WebApi

ASP.NET 4.6.2 Templates

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Learn more

Add folders and core references for:

☐ Web Forms ☑ MVC ☑ Web API

☐ Enable Docker Compose support (Requires Docker for Windows)

☐ Add unit tests

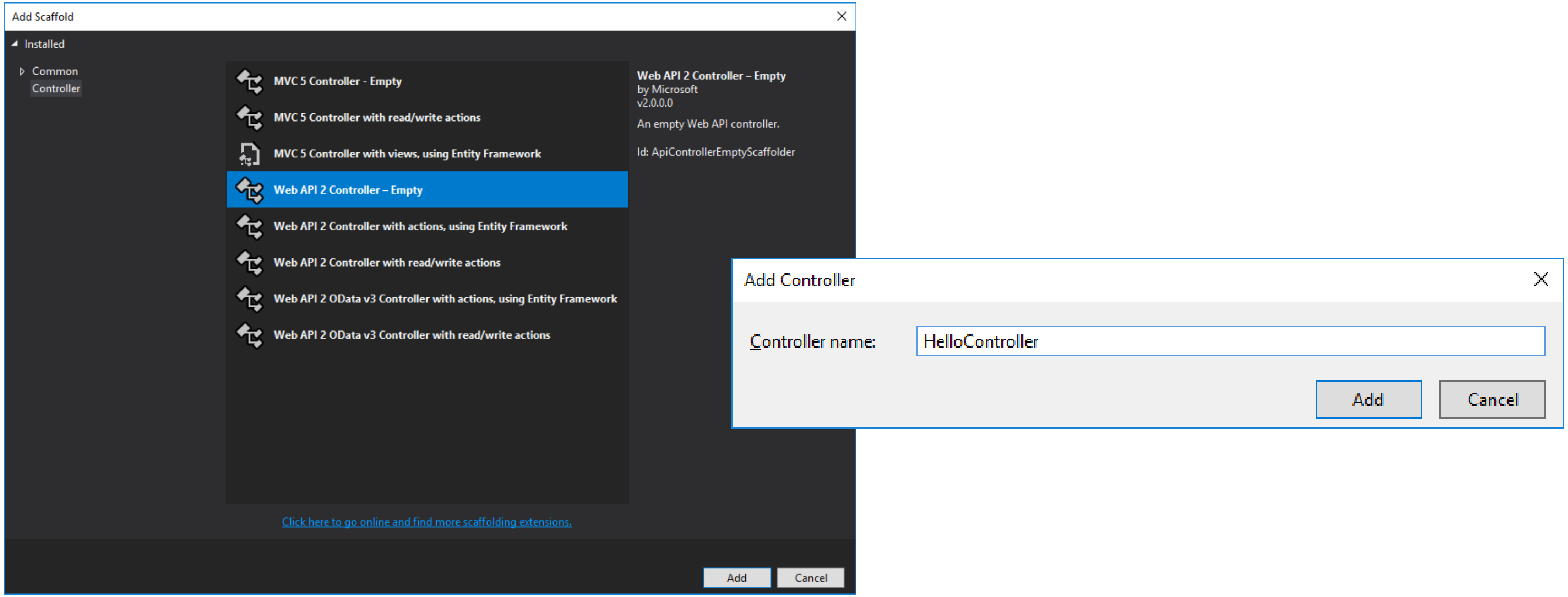
Test project name: my_WebApi.Tests

Authentication: **No Authentication**

Create Web API project

Add Web API controller by right clicking on the

Controllers folder → Controller.



Add Scaffold

- Installed
  - Common
  - Controller

MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

Web API 2 Controller – Empty
by Microsoft
v2.0.0.0
An empty Web API controller.
Id: ApiControllerEmptyScaffolder

Click here to go online and find more scaffolding extensions.

Add | Cancel

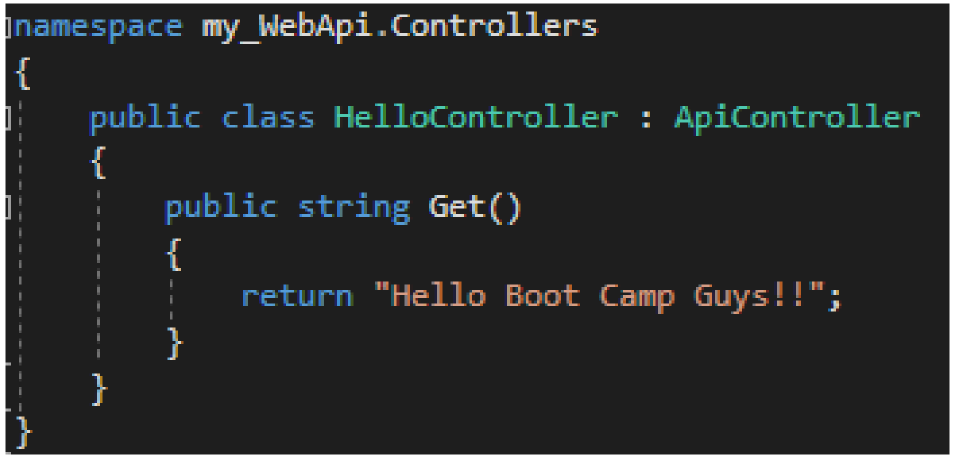
Add Controller

Controller name: HelloController

Add | Cancel

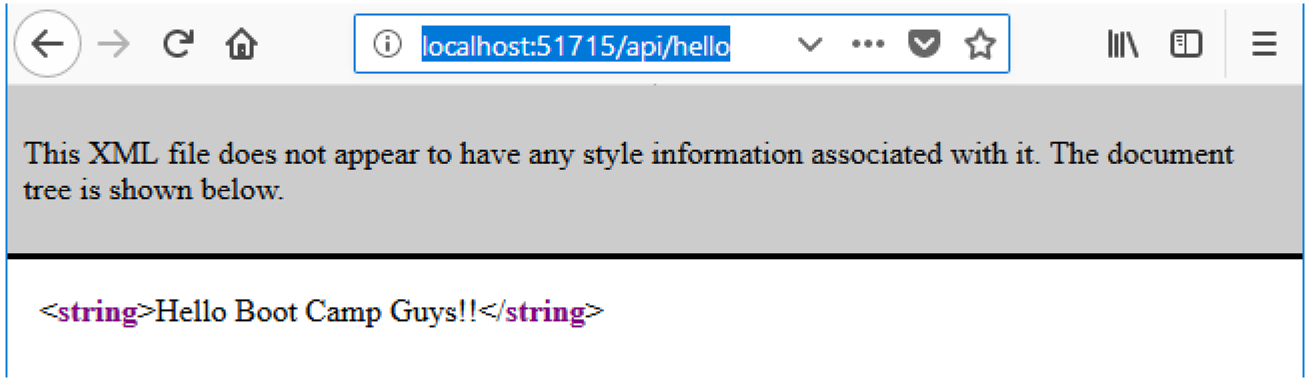
Create Web API project

Add simple Get action method



```
namespace my_WebApi.Controllers
{
    public class HelloController : ApiController
    {
        public string Get()
        {
            return "Hello Boot Camp Guys!!";
        }
    }
}
```

Compile and run the project and navigate to */api/hello*



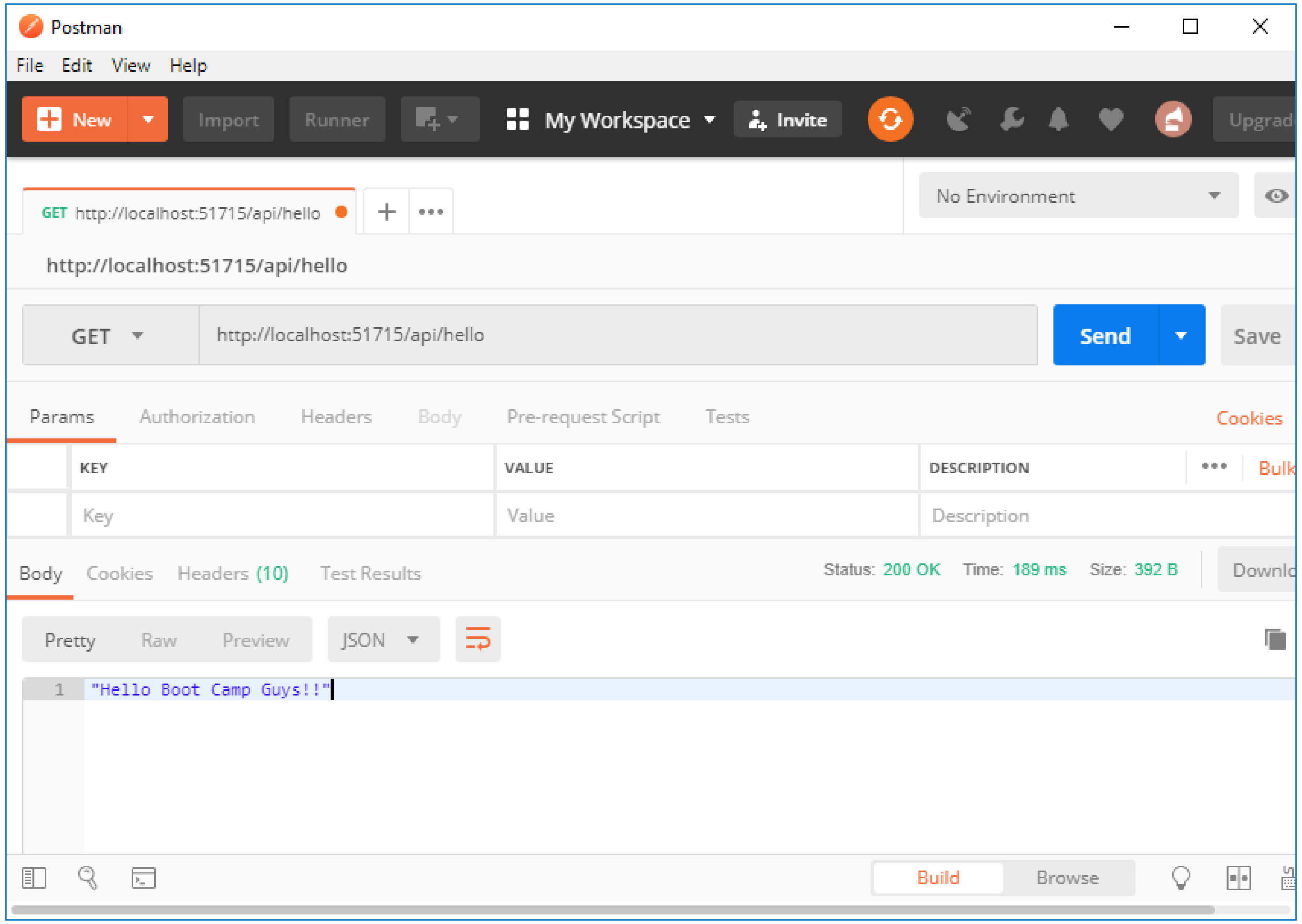
localhost:51715/api/hello

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string>Hello Boot Camp Guys!!</string>
```

Test Web API

Test you Web API using Postman



GET http://localhost:51715/api/hello

http://localhost:51715/api/hello

GET http://localhost:51715/api/hello

KEY	VALUE	DESCRIPTION
Key	Value	Description

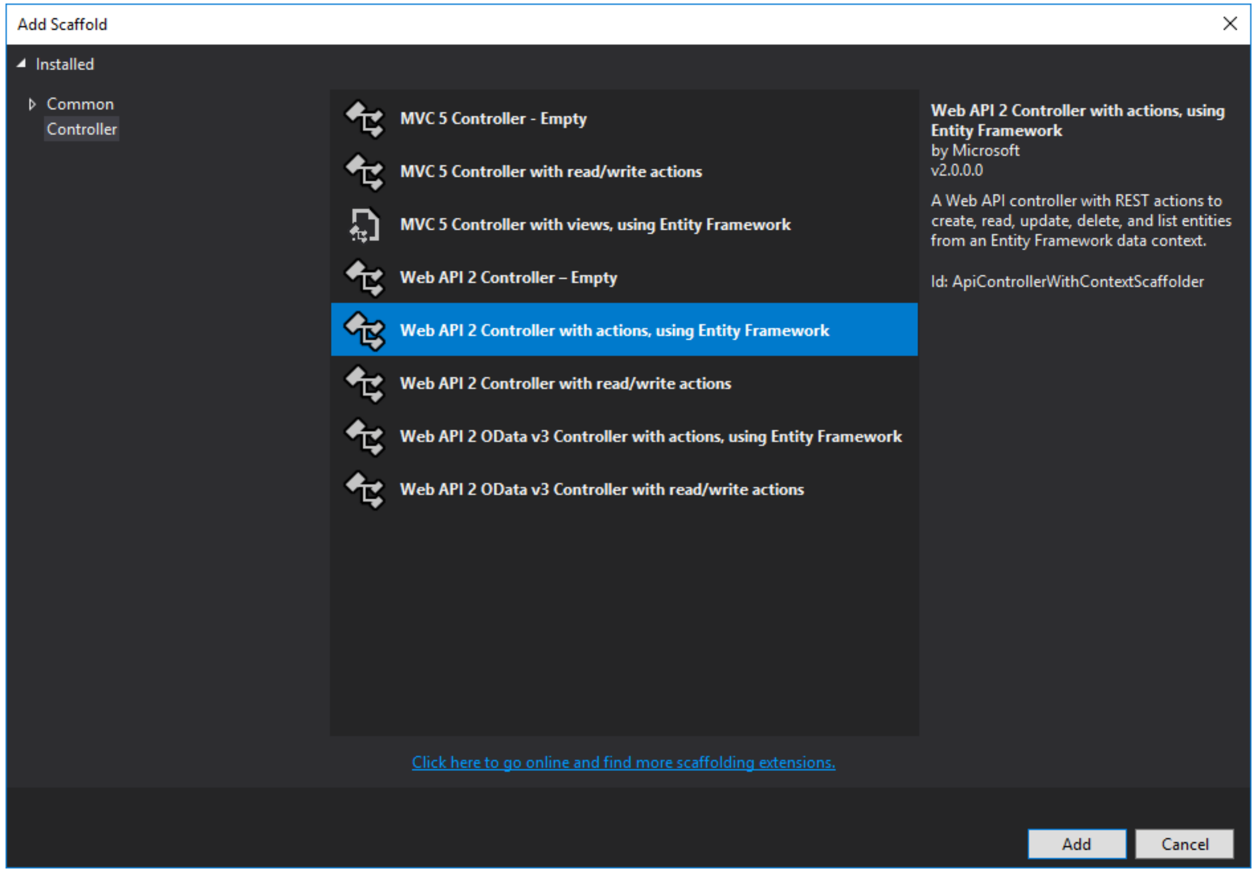
Status: 200 OK Time: 189 ms Size: 392 B

```
"Hello Boot Camp Guys!!"
```

API Controllers

Web API Controller is similar to ASP.NET MVC controller. It handles incoming HTTP requests and send response back to the caller.

Lets create a Web API controller using Entity Framework.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
Click here to go online and find more scaffolding extensions.
Add
Cancel

Web API 2 Controller with actions, using Entity Framework
by Microsoft
v2.0.0.0

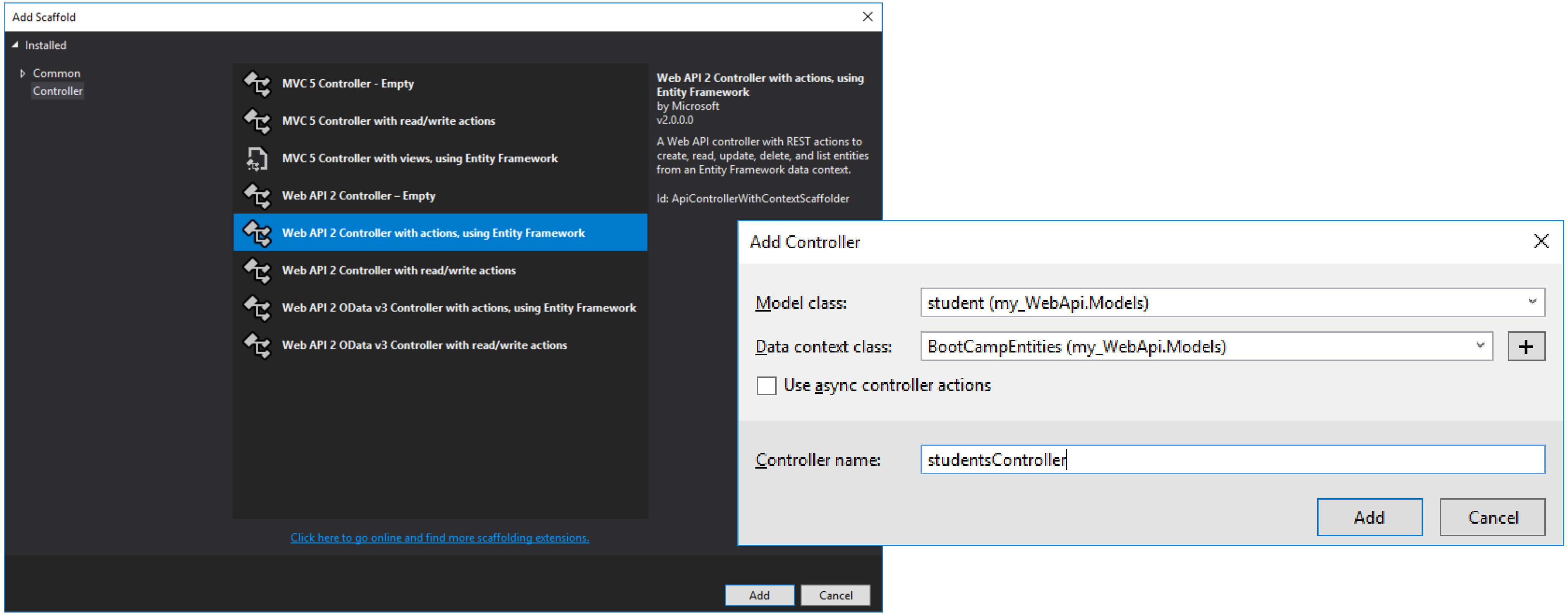
A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

Id: ApiControllerWithContextScaffolder

API Controllers

Web API Controller is similar to ASP.NET MVC controller. It handles incoming HTTP requests and send response back to the caller.

Lets create a Web API controller using Entity Framework.



Add Scaffold

Installed
- Common
- Controller

- MVC 5 Controller - Empty
- MVC 5 Controller with read/write actions
- MVC 5 Controller with views, using Entity Framework
- Web API 2 Controller – Empty
- Web API 2 Controller with actions, using Entity Framework
- Web API 2 Controller with read/write actions
- Web API 2 OData v3 Controller with actions, using Entity Framework
- Web API 2 OData v3 Controller with read/write actions

Web API 2 Controller with actions, using Entity Framework
by Microsoft
v2.0.0.0

A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

Id: ApiControllerWithContextScaffolder

Click here to go online and find more scaffolding extensions.

Add Cancel

Add Controller

Model class: student (my_WebApi.Models)

Data context class: BootCampEntities (my_WebApi.Models)

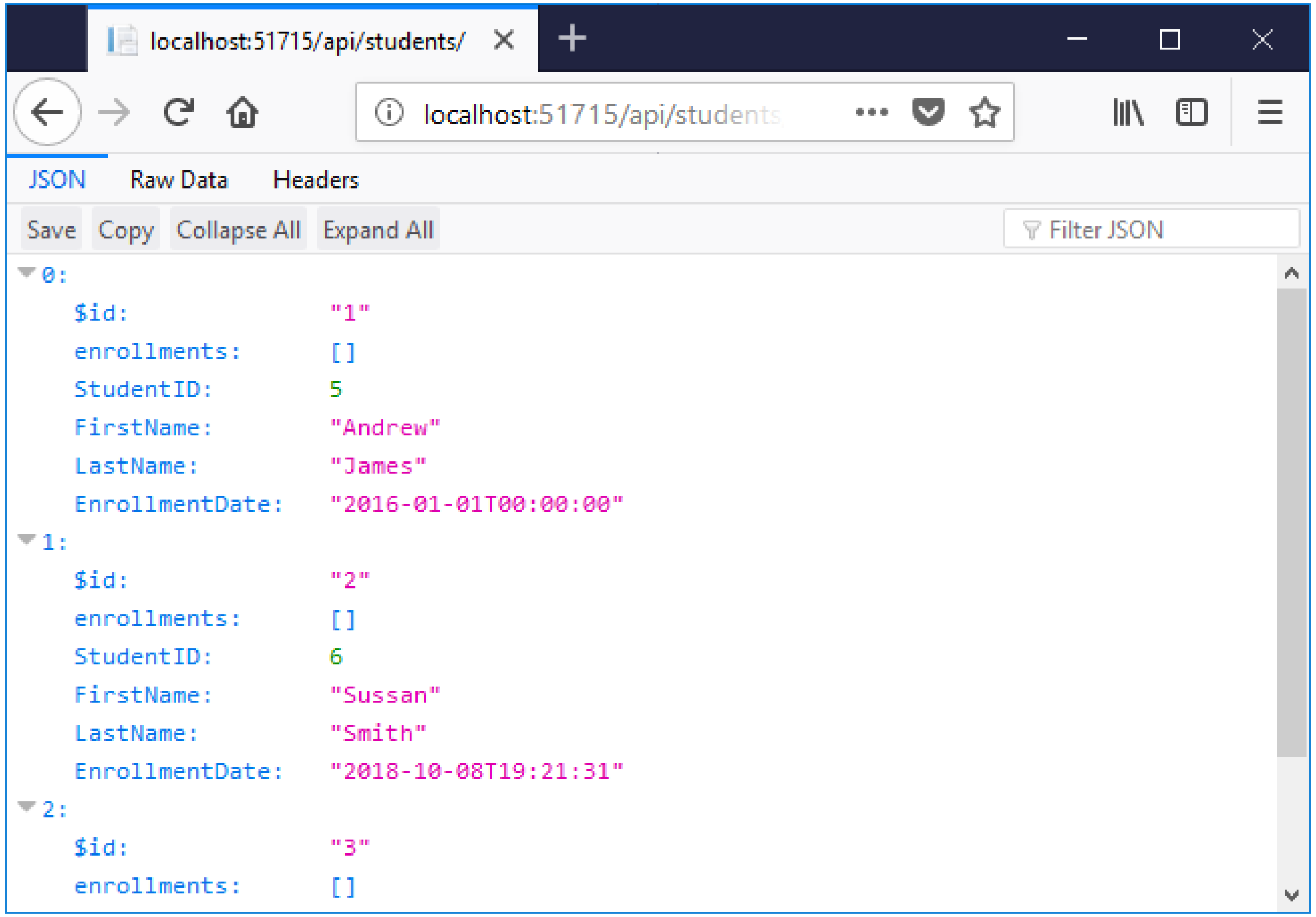
☐ Use async controller actions

Controller name: studentsController

Add Cancel

API Controllers

Compile and run the project and navigate to */api/students/*


localhost:51715/api/students/

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
0:
    $id:              "1"
    enrollments:      []
    StudentID:        5
    FirstName:        "Andrew"
    LastName:         "James"
    EnrollmentDate:   "2016-01-01T00:00:00"
1:
    $id:              "2"
    enrollments:      []
    StudentID:        6
    FirstName:        "Sussan"
    LastName:         "Smith"
    EnrollmentDate:   "2018-10-08T19:21:31"
2:
    $id:              "3"
    enrollments:      []
```

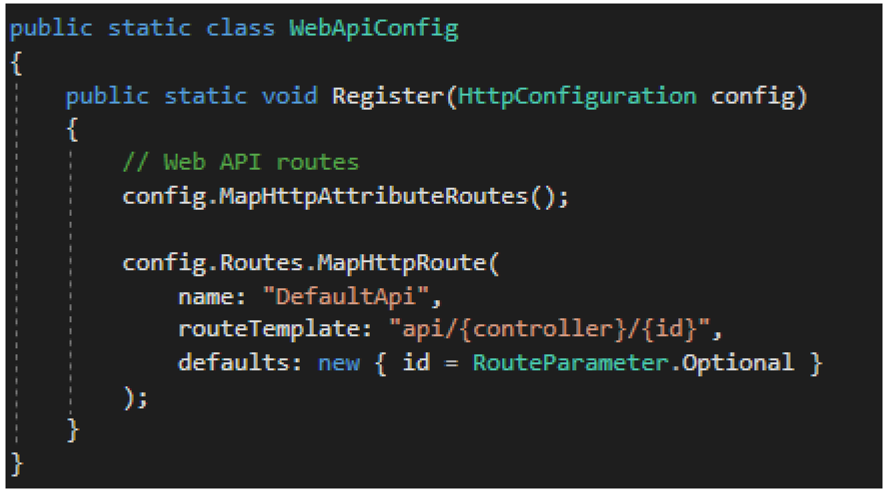
API Routing

API Routing, routes an incoming HTTP request to a particular action method on a Web API controller.
Web API supports two types of routing:

1. Convention-based Routing
2. Attribute Routing

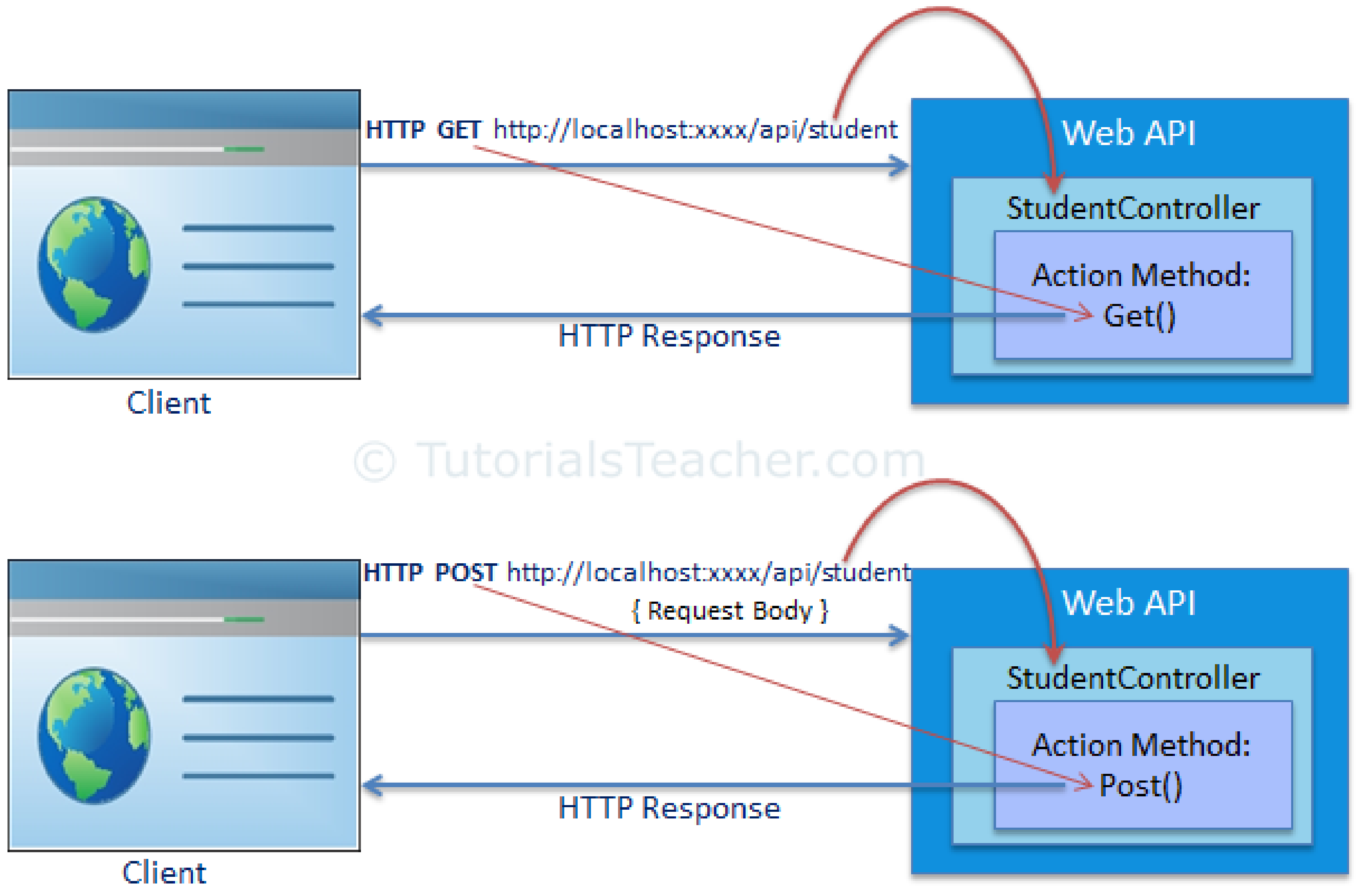
Convention-based Routing

In the convention-based routing, Web API uses route templates to determine which controller and action method to execute. At least one route template must be added into route table in order to handle various HTTP requests.



```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

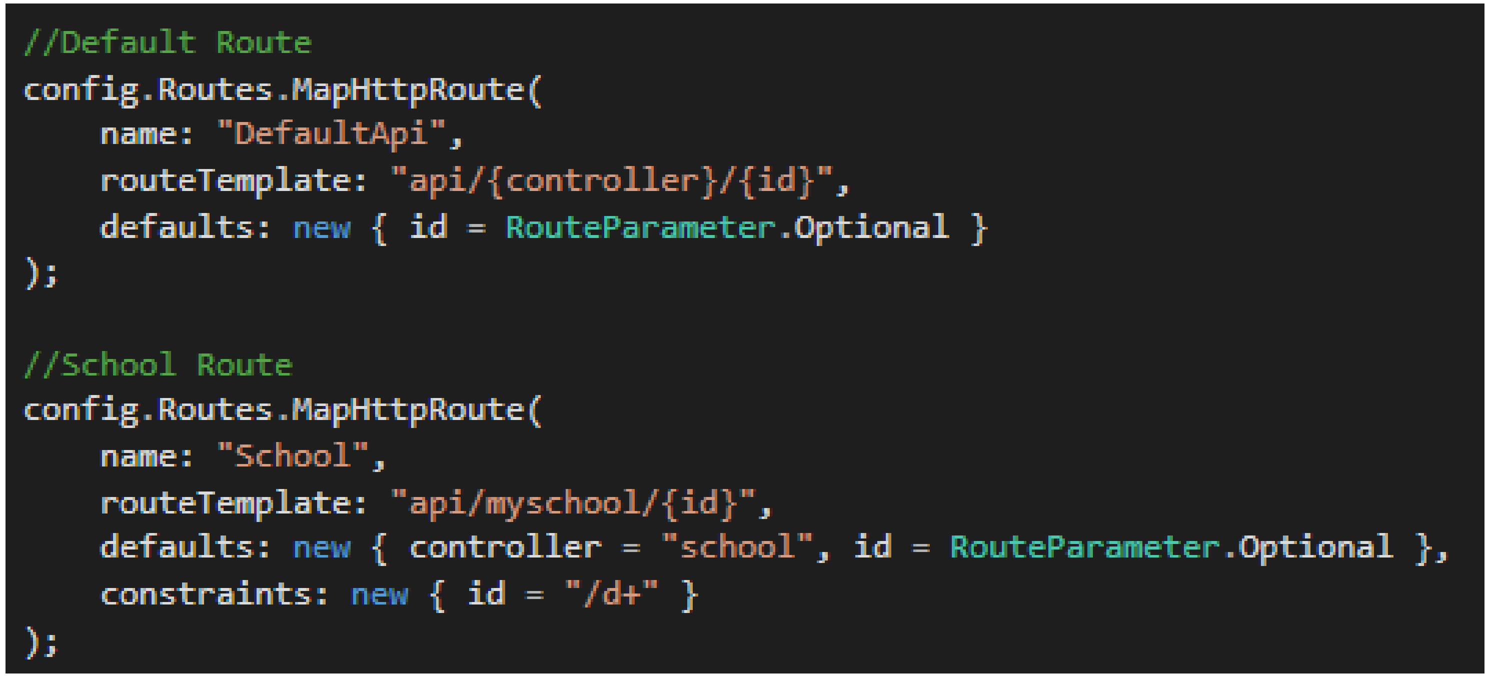
API Routing


HTTP GET http://localhost:xxxx/api/student
Web API
StudentController
Action Method:
Get()
HTTP Response
Client
© TutorialsTeacher.com
HTTP POST http://localhost:xxxx/api/student
{ Request Body }
Web API
StudentController
Action Method:
Post()
HTTP Response
Client

API Routing

Configure Multiple Routes

We configured a single route above. However, you can configure multiple routes in the Web API using HttpConfiguration object. The following example demonstrates configuring multiple routes.

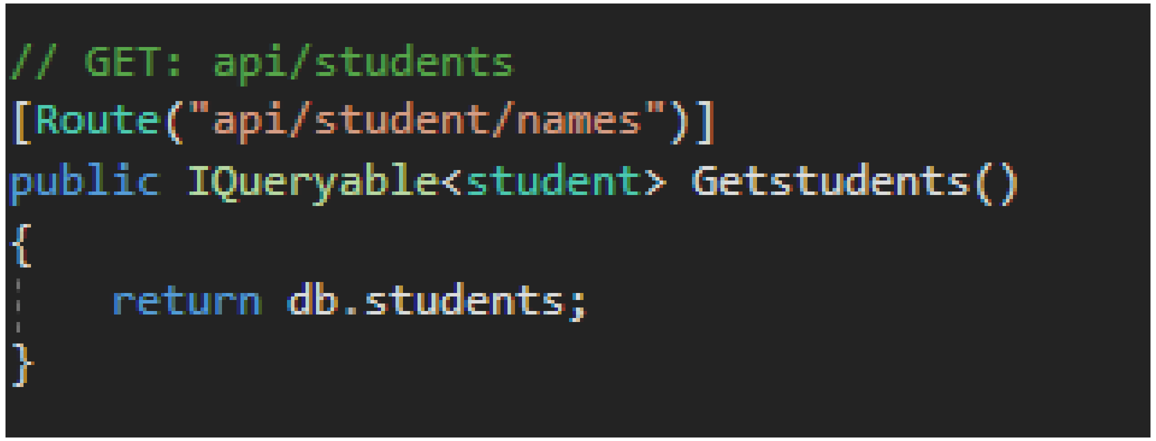


```
//Default Route
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);

//School Route
config.Routes.MapHttpRoute(
    name: "School",
    routeTemplate: "api/myschool/{id}",
    defaults: new { controller = "school", id = RouteParameter.Optional },
    constraints: new { id = "/d+" }
);
```

API Routing

Attribute Routing

Attribute routing is supported in Web API 2. As the name implies, attribute routing uses [Route()] attribute to define routes. The *Route* attribute can be applied on any controller or action method.



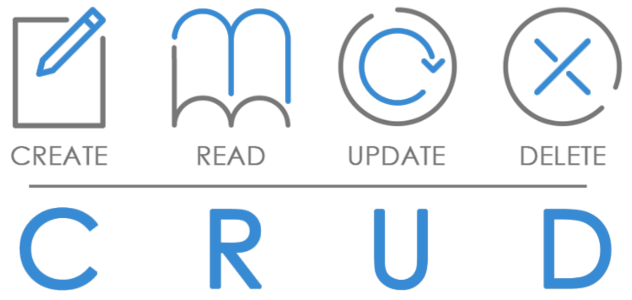
```
// GET: api/students
[Route("api/student/names")]
public IQueryable<student> Getstudents()
{
    return db.students;
}
```

In this example the *Route* attribute defines new route "api/student/names" which will be handled by the Get() action method of StudentController.

HTTP Verbs

The four main HTTP methods (GET, PUT, POST, and DELETE) can be mapped to CRUD operations as follows:

- **GET** retrieves the representation of the resource at a specified URI. GET should have no side effects on the server.
- **PUT** updates a resource at a specified URI. PUT can also be used to create a new resource at a specified URI, if the server allows clients to specify new URIs.
- **POST** creates a new resource. The server assigns the URI for the new object and returns this URI as part of the response message.
- **DELETE** deletes a resource at a specified URI.

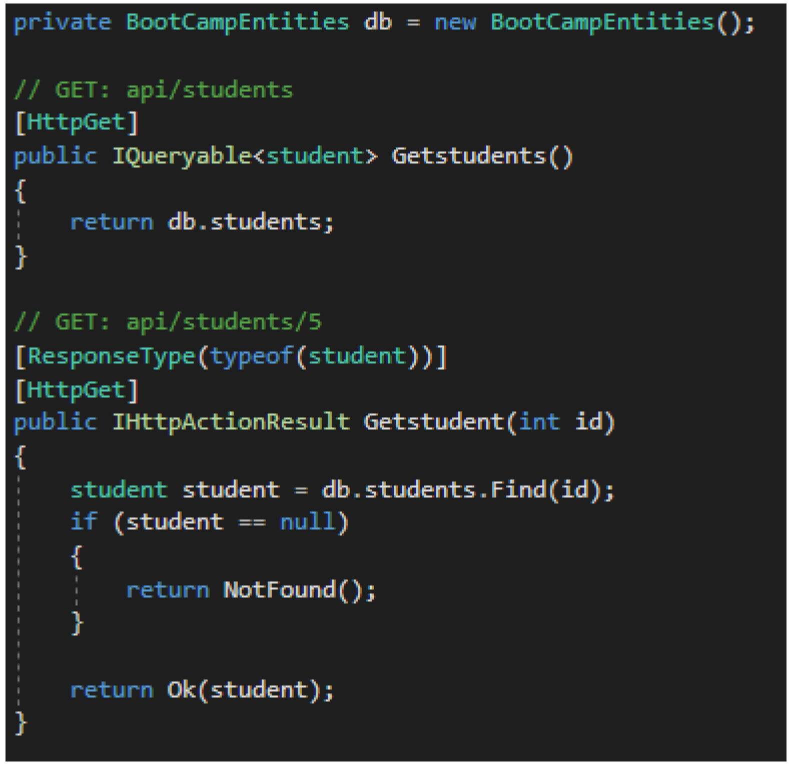


HTTP Verbs

HTTP GET

- This verb should be used only to get information or data from database or other source.

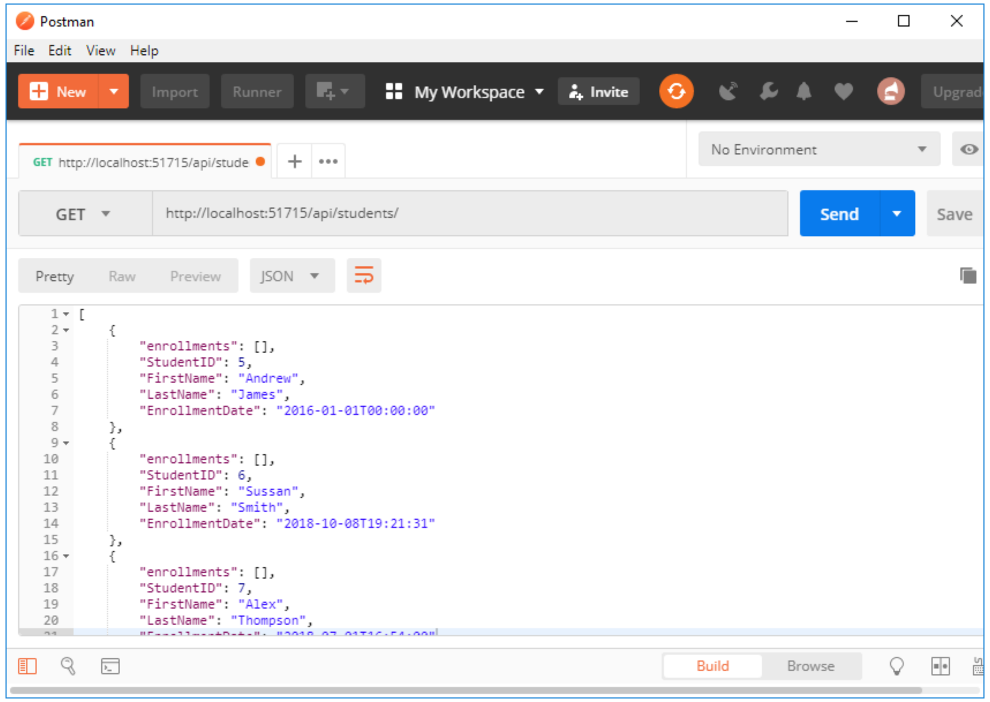
- Action method that starts with a work "Get" will handle HTTP GET request. We can either name it only Get or with any suffix. Let's add our first Get action method and give it a name GetAllStudents because it will return all the students from the DB.



```
private BootCampEntities db = new BootCampEntities();

// GET: api/students
[HttpGet]
public IQueryable<student> Getstudents()
{
    return db.students;
}

// GET: api/students/5
[ResponseType(typeof(student))]
[HttpGet]
public IHttpActionResult Getstudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        return NotFound();
    }

    return Ok(student);
}
```

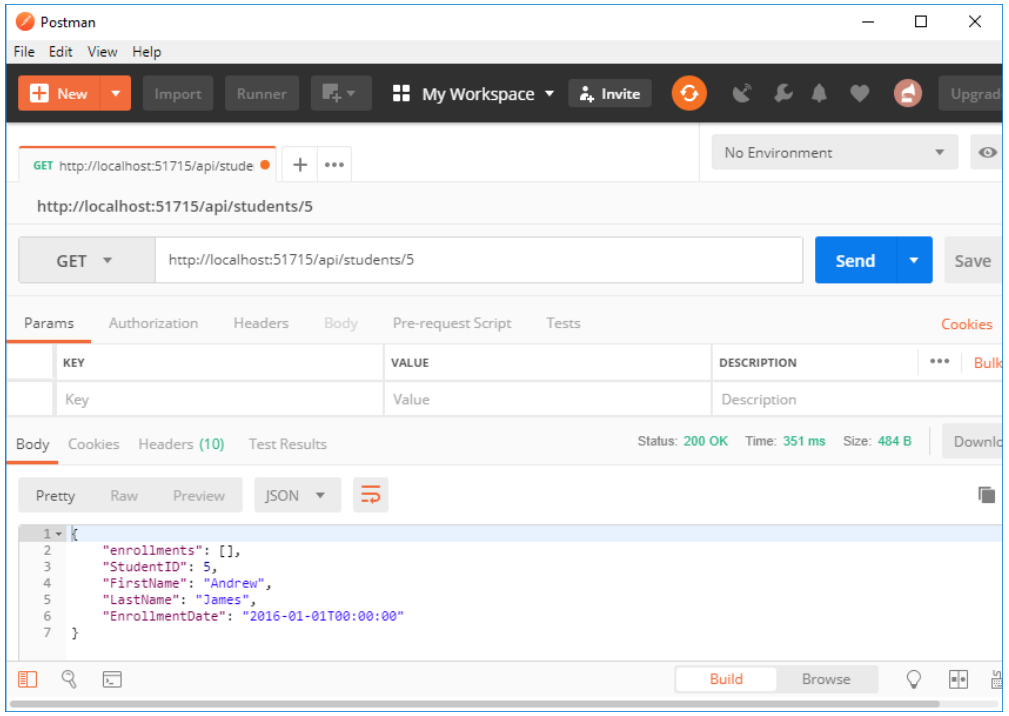
HTTP Verbs

HTTP GET



GET http://localhost:51715/api/students/

```
[
    {
        "enrollments": [],
        "StudentID": 5,
        "FirstName": "Andrew",
        "LastName": "James",
        "EnrollmentDate": "2016-01-01T00:00:00"
    },
    {
        "enrollments": [],
        "StudentID": 6,
        "FirstName": "Sussan",
        "LastName": "Smith",
        "EnrollmentDate": "2018-10-08T19:21:31"
    },
    {
        "enrollments": [],
        "StudentID": 7,
        "FirstName": "Alex",
        "LastName": "Thompson",
```



http://localhost:51715/api/students/5

GET http://localhost:51715/api/students/5

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 351 ms Size: 484 B

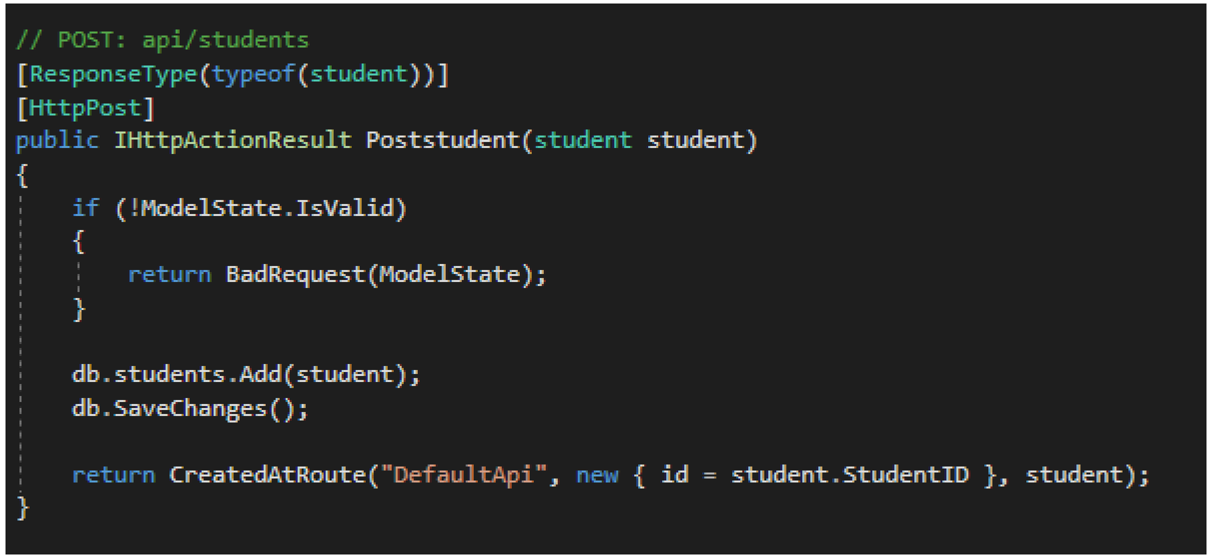
```
{
    "enrollments": [],
    "StudentID": 5,
    "FirstName": "Andrew",
    "LastName": "James",
    "EnrollmentDate": "2016-01-01T00:00:00"
}
```

HTTP Verbs

HTTP POST

- The HTTP POST request is used to create a new record in the data source in the RESTful architecture.

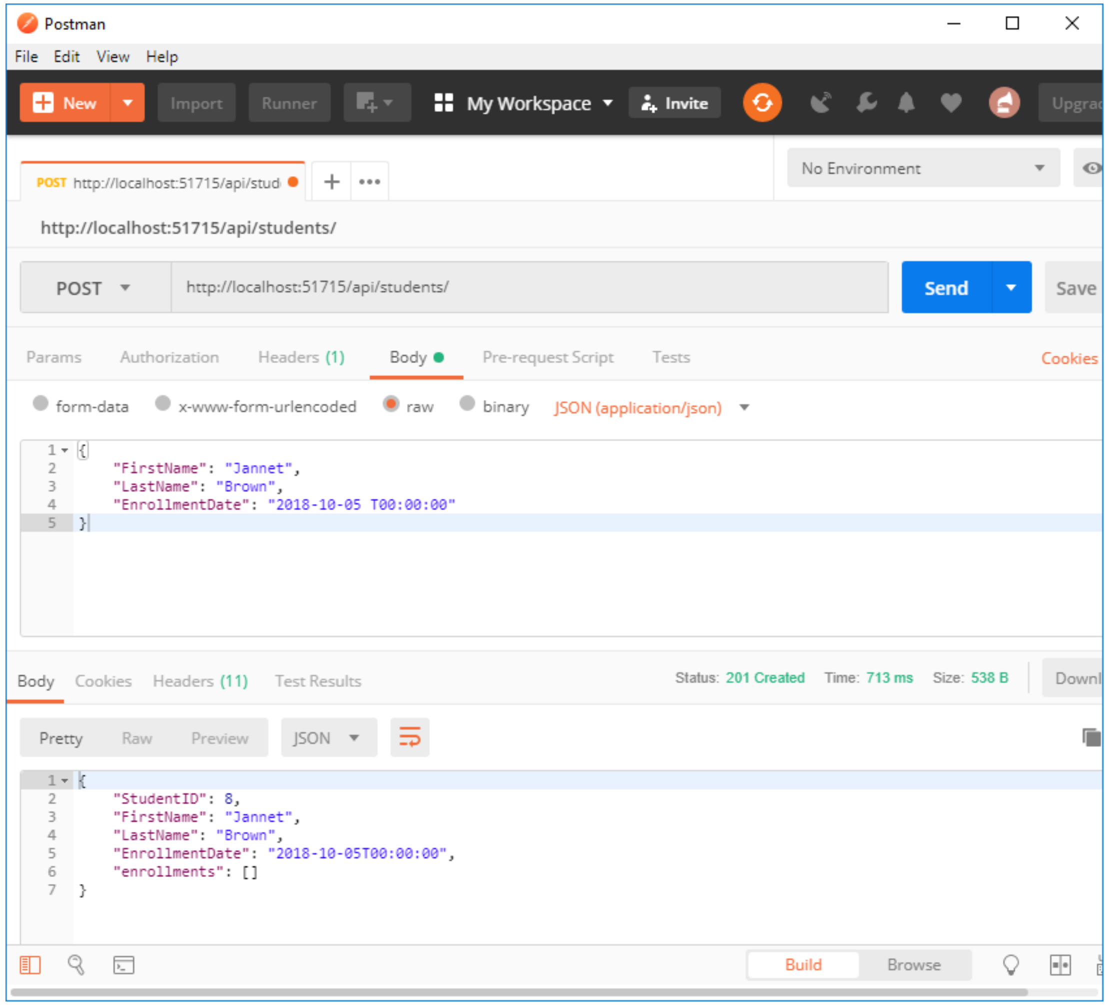
- The action method that will handle HTTP POST request must start with a word Post. It can be named either Post or with any suffix e.g. POST(), Post(), PostNewStudent(), PostStudents() are valid names for an action method that handles HTTP POST request.



```
// POST: api/students
[ResponseType(typeof(student))]
[HttpPost]
public IHttpActionResult Poststudent(student student)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.students.Add(student);
    db.SaveChanges();

    return CreatedAtRoute("DefaultApi", new { id = student.StudentID }, student);
}
```

HTTP Verbs

HTTP POST



http://localhost:51715/api/students/

POST http://localhost:51715/api/students/

[✓ raw] JSON (application/json)

```
{
    "FirstName": "Jannet",
    "LastName": "Brown",
    "EnrollmentDate": "2018-10-05 T00:00:00"
}
```

Status: 201 Created Time: 713 ms Size: 538 B

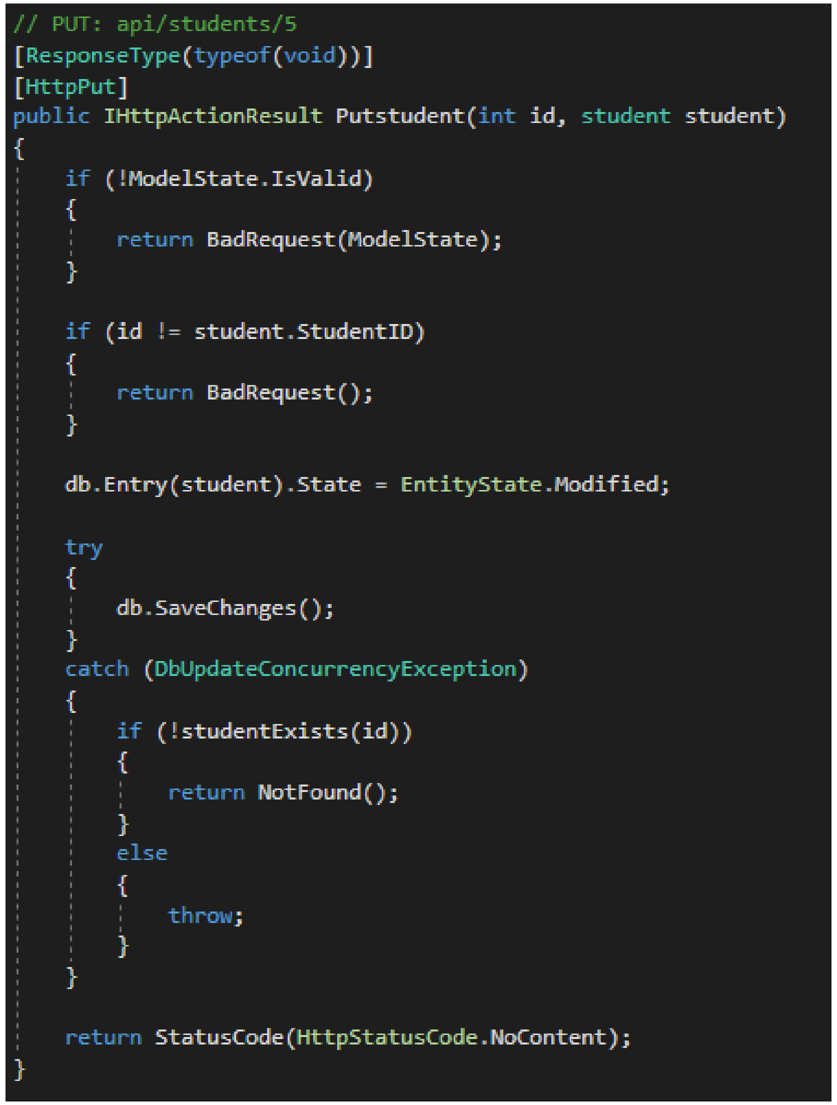
```
{
    "StudentID": 8,
    "FirstName": "Jannet",
    "LastName": "Brown",
    "EnrollmentDate": "2018-10-05T00:00:00",
    "enrollments": []
}
```

HTTP Verbs

HTTP PUT

- The HTTP PUT method is used to update an existing record in the data source in the RESTful architecture.

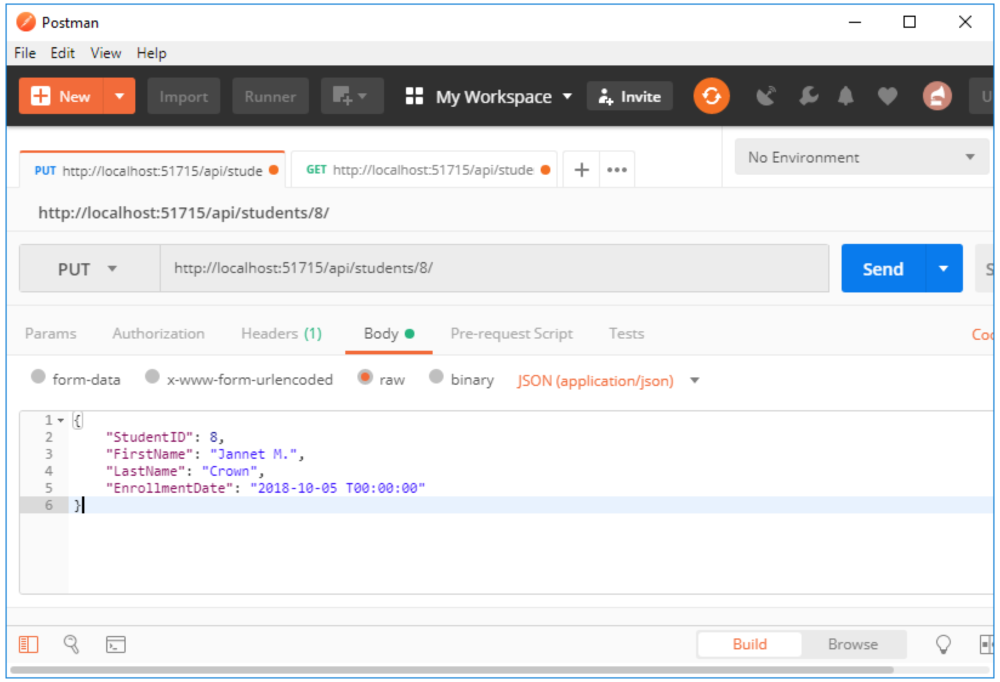
- The action method that will handle HTTP PUT request must start with a word Put. It can be named either Put or with any suffix e.g. PUT(), Put(), PutStudent(), PutStudents() are valid names for an action method that handles HTTP PUT request.



```
// PUT: api/students/5
[ResponseType(typeof(void))]
[HttpPut]
public IHttpActionResult Putstudent(int id, student student)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != student.StudentID)
    {
        return BadRequest();
    }

    db.Entry(student).State = EntityState.Modified;

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!studentExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return StatusCode(HttpStatusCode.NoContent);
}
```

HTTP Verbs

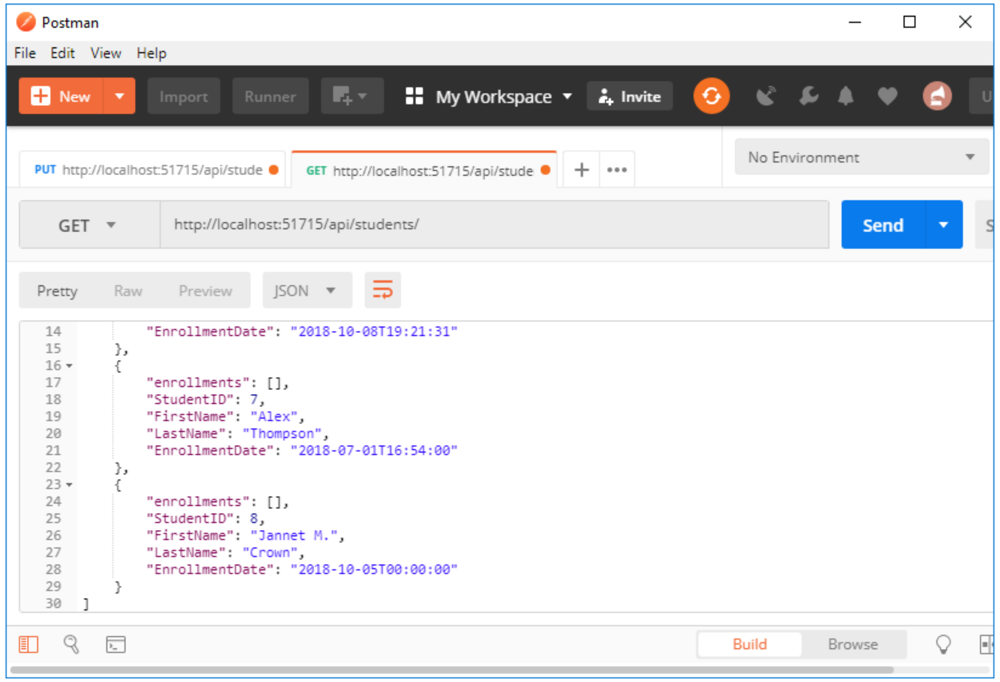
HTTP PUT



PUT http://localhost:51715/api/students/8/

Body: raw, JSON (application/json)

```
{
    "StudentID": 8,
    "FirstName": "Jannet M.",
    "LastName": "Crown",
    "EnrollmentDate": "2018-10-05 T00:00:00"
}
```



GET http://localhost:51715/api/students/

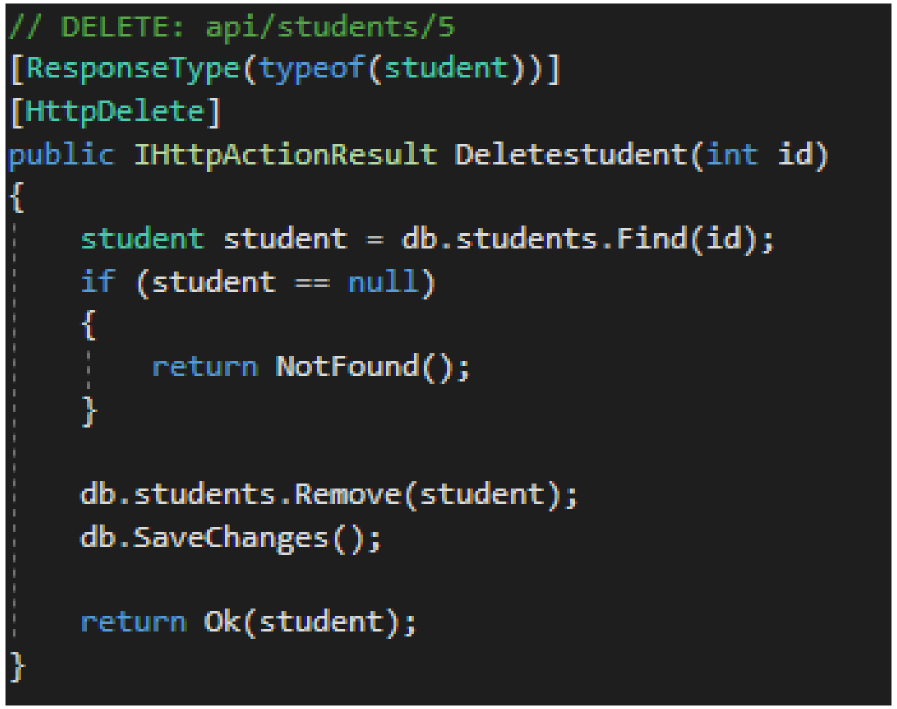
```
        "EnrollmentDate": "2018-10-08T19:21:31"
    },
    {
        "enrollments": [],
        "StudentID": 7,
        "FirstName": "Alex",
        "LastName": "Thompson",
        "EnrollmentDate": "2018-07-01T16:54:00"
    },
    {
        "enrollments": [],
        "StudentID": 8,
        "FirstName": "Jannet M.",
        "LastName": "Crown",
        "EnrollmentDate": "2018-10-05T00:00:00"
    }
]
```

HTTP Verbs

HTTP DELETE

- The HTTP DELETE request is used to delete an existing record in the data source in the RESTful architecture.

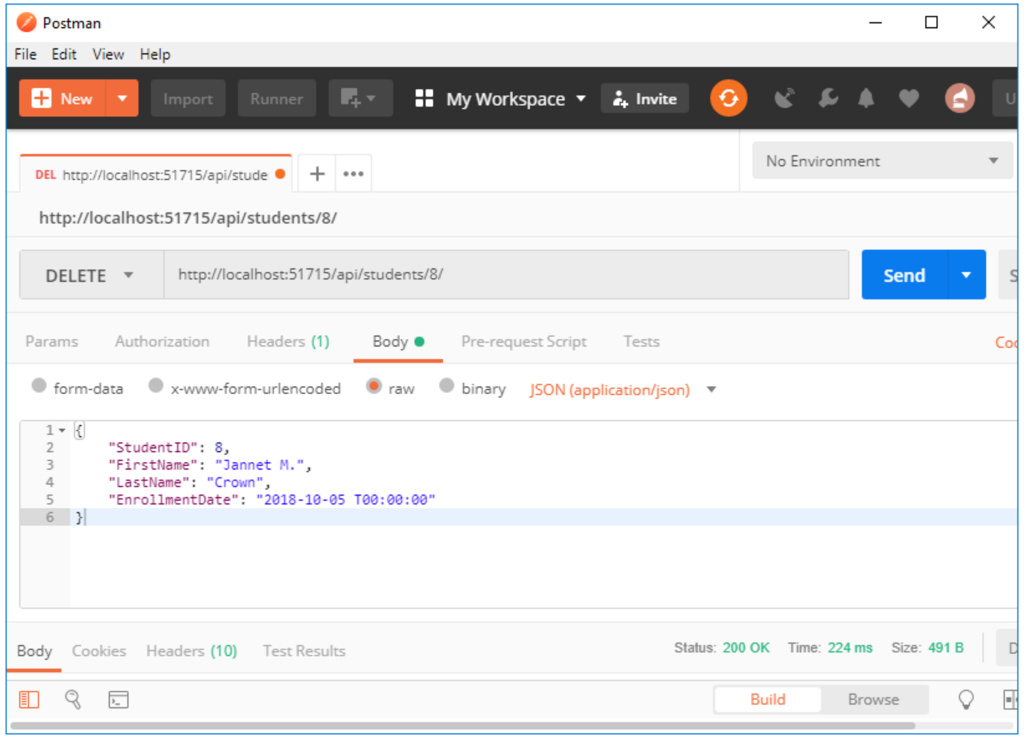
- The action method that will handle HTTP DELETE request must start with the word "Delete". It can be named either Delete or with any suffix e.g. DELETE(), Delete(), DeleteStudent(), DeleteAllStudents() are valid names for an action method that handles HTTP DELETE request.



```
// DELETE: api/students/5
[ResponseType(typeof(student))]
[HttpDelete]
public IHttpActionResult Deletestudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        return NotFound();
    }

    db.students.Remove(student);
    db.SaveChanges();

    return Ok(student);
}
```

HTTP Verbs

HTTP DELETE


Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
No Environment
DEL http://localhost:51715/api/stude
Params Authorization Headers (1) Body Pre-request Script Tests
form-data x-www-form-urlencoded raw binary
Body Cookies Headers (10) Test Results
Build Browse

http://localhost:51715/api/students/8/

DELETE http://localhost:51715/api/students/8/ Send

Body: raw, JSON (application/json)

```
{
    "StudentID": 8,
    "FirstName": "Jannet M.",
    "LastName": "Crown",
    "EnrollmentDate": "2018-10-05 T00:00:00"
}
```

Status: 200 OK Time: 224 ms Size: 491 B

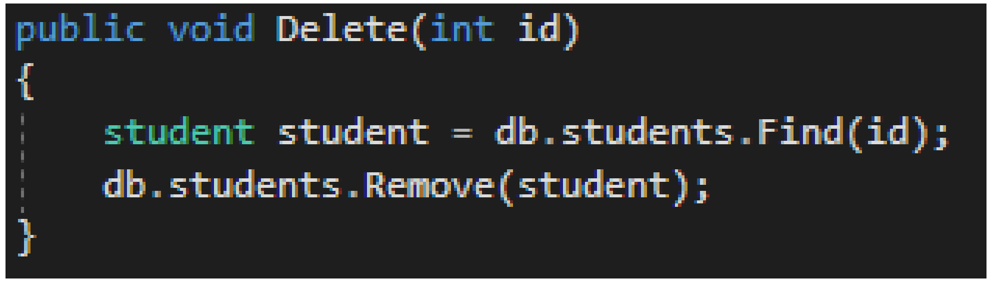
Web API Result Types

The Web API action method can have following return types:

- *Void*
- *Primitive type or Complex type*
- *HttpResponseMessage*
- *IHttpActionResult*

Void

It's not necessary that all action methods must return something. It can have void return type.

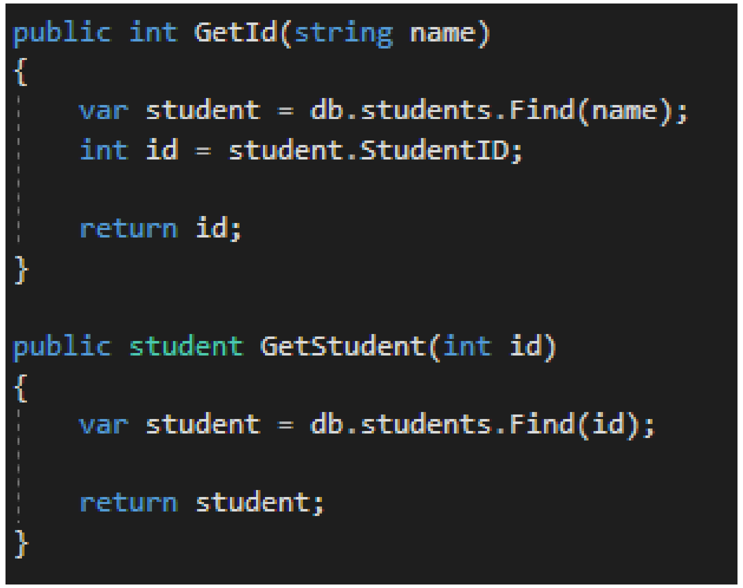


```
public void Delete(int id)
{
    student student = db.students.Find(id);
    db.students.Remove(student);
}
```

Web API Result Types

Primitive or Complex Type

An action method can return primitive or other custom complex types as other normal methods.

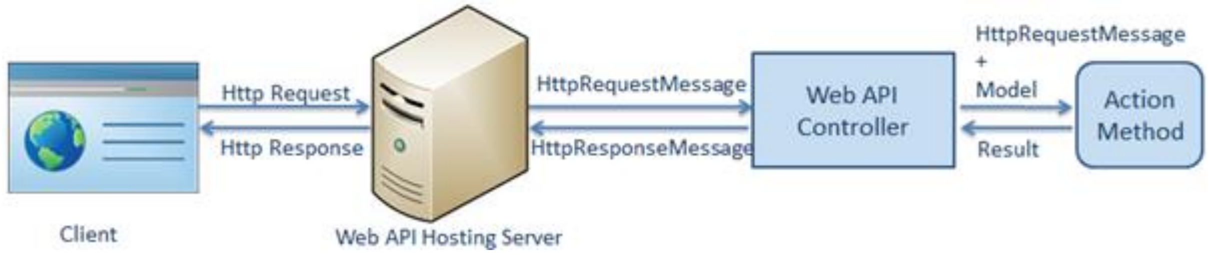


```
public int GetId(string name)
{
    var student = db.students.Find(name);
    int id = student.StudentID;

    return id;
}

public student GetStudent(int id)
{
    var student = db.students.Find(id);

    return student;
}
```

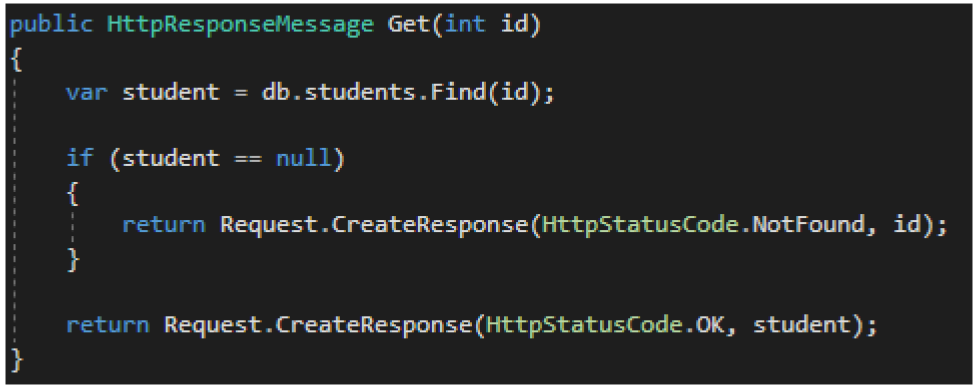
Web API Result Types

HttpResponseMessage

Web API controller always returns an object of HttpResponseMessage to the hosting infrastructure.



The advantage of sending HttpResponseMessage from an action method is that you can configure a response your way. You can set the status code, content or error message (if any) as per your requirement.

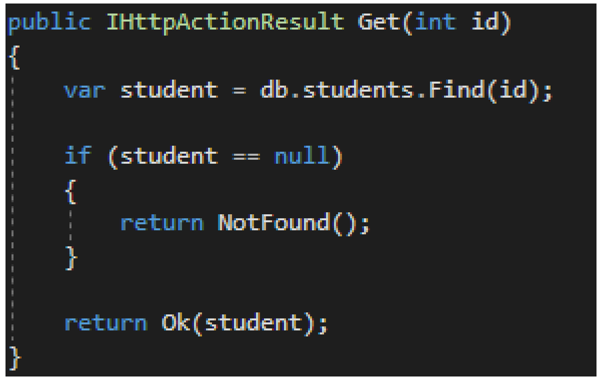


```
public HttpResponseMessage Get(int id)
{
    var student = db.students.Find(id);

    if (student == null)
    {
        return Request.CreateResponse(HttpStatusCode.NotFound, id);
    }

    return Request.CreateResponse(HttpStatusCode.OK, student);
}
```

Web API Result Types

IHttpActionResult

The *IHttpActionResult* was introduced in Web API 2 (.NET 4.5). An action method in Web API 2 can return an implementation of IHttpActionResult class which is more or less similar to ActionResult class in ASP.NET MVC.
You can create your own class that implements IHttpActionResult or use various methods of ApiController class that returns an object that implement the IHttpActionResult.



```
public IHttpActionResult Get(int id)
{
    var student = db.students.Find(id);

    if (student == null)
    {
        return NotFound();
    }

    return Ok(student);
}
```

ApiController Method	Description
BadRequest()	Creates a BadRequestResult object with status code 400.
Conflict()	Creates a ConflictResult object with status code 409.
Content()	Creates a NegotiatedContentResult with the specified status code and data.
Created()	Creates a CreatedNegotiatedContentResult with status code 201 Created.
CreatedAtRoute()	Creates a CreatedAtRouteNegotiatedContentResult with status code 201 created.
InternalServerError()	Creates an InternalServerErrorResult with status code 500 Internal server error.
NotFound()	Creates a NotFoundResult with status code404.
Ok()	Creates an OkResult with status code 200.
Redirect()	Creates a RedirectResult with status code 302.
RedirectToRoute()	Creates a RedirectToRouteResult with status code 302.
ResponseMessage()	Creates a ResponseMessageResult with the specified HttpResponseMessage.
StatusCode()	Creates a StatusCodeResult with the specified http status code.
Unauthorized()	Creates an UnauthorizedResult with status code 401.

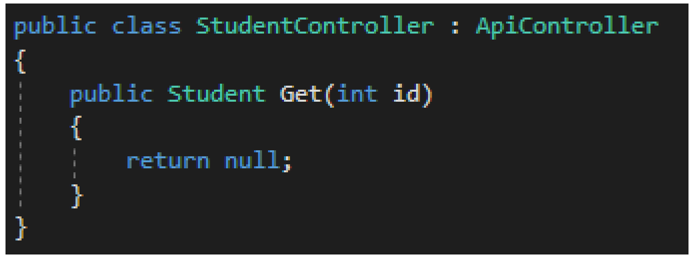
Parameter Binding

Action methods in Web API controller can have one or more parameters of different types. It can be either primitive type or complex type. Web API binds action method parameters either with URL's query string or with request body depending on the parameter type. By default, if parameter type is of .NET primitive type such as int, bool, double, string, GUID, DateTime, decimal or any other type that can be converted from string type then it sets the value of a parameter from the query string. And if the parameter type is complex type then Web API tries to get the value from request body by default.

HTTP Method	Query String	Request Body
GET	Primitive Type, Complex Type	NA
POST	Primitive Type	Complex Type
PUT	Primitive Type	Complex Type
DELETE	Primitive Type, Complex Type	NA

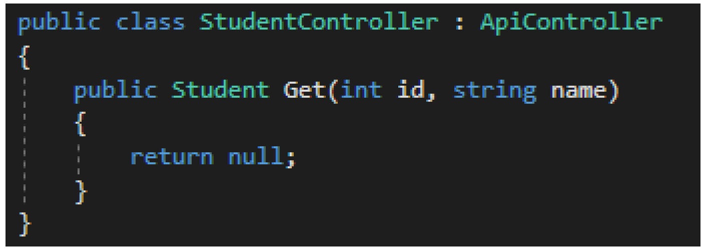
Parameter Binding

Get Action Method with Primitive Parameter



```
public class StudentController : ApiController
{
    public Student Get(int id)
    {
        return null;
    }
}
```

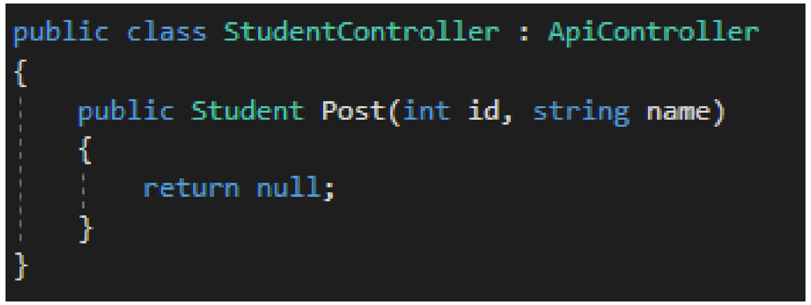
Get Action Method with Multiple Primitive Parameters



```
public class StudentController : ApiController
{
    public Student Get(int id, string name)
    {
        return null;
    }
}
```

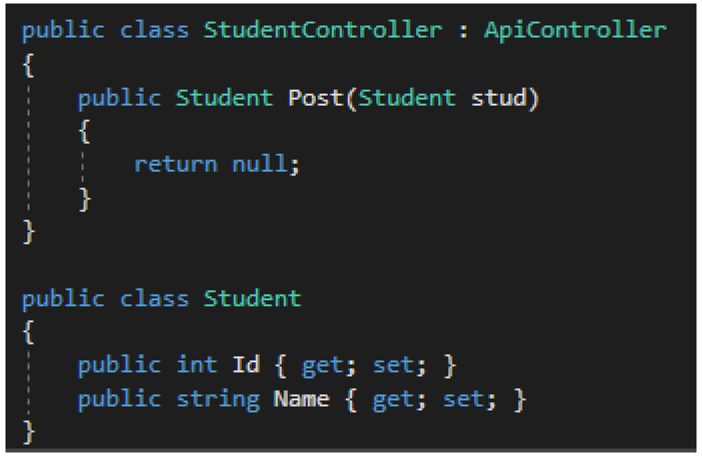
Parameter Binding

Post Action Method with Primitive Parameter



```
public class StudentController : ApiController
{
    public Student Post(int id, string name)
    {
        return null;
    }
}
```

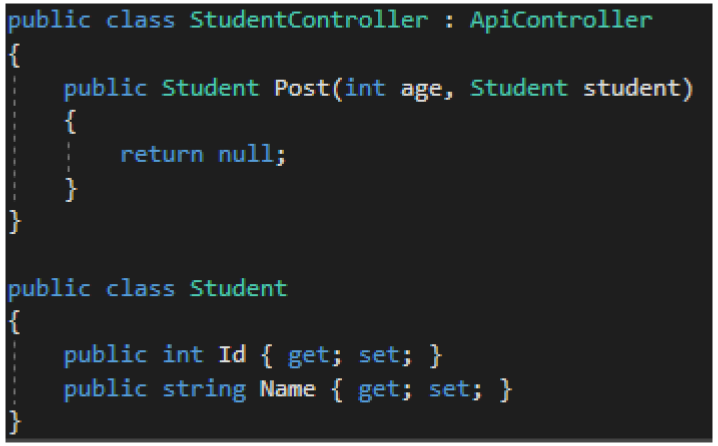
Post Action Method with Post Method with Complex Type Parameter



```
public class StudentController : ApiController
{
    public Student Post(Student stud)
    {
        return null;
    }
}

public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Parameter Binding

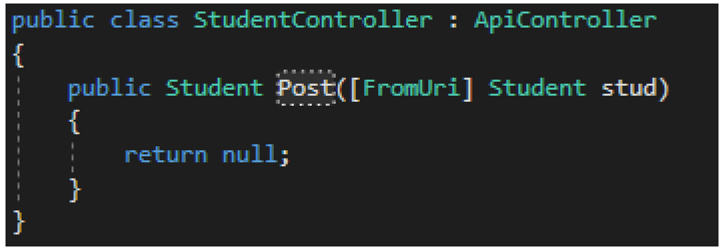
Post Method with Mixed Parameters



```
public class StudentController : ApiController
{
    public Student Post(int age, Student student)
    {
        return null;
    }
}

public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Post FromUri

- *http://localhost:xxxx/api/student?id=1&name=steve*



```
public class StudentController : ApiController
{
    public Student Post([FromUri] Student stud)
    {
        return null;
    }
}
```

What is Cors?

CORS stands for Cross-Origin Resource Sharing. It is a mechanism that allows restricted resources on a web page to be requested from another domain, outside the domain from which the resource originated. A web page may freely embed images, stylesheets, scripts, iframes, and videos.

For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. For example, XMLHttpRequest follows the same-origin policy. So, a web application using XMLHttpRequest could only make HTTP requests to its own domain. To improve web applications, developers asked browser vendors to allow XMLHttpRequest to make cross-domain requests.

What is "Same Origin"?
Two URLs have the same origin if they have identical schemes, hosts, and ports.

- http://example.com/foo.html
- http://example.com/bar.html

These URLs have different origins than the previous two:

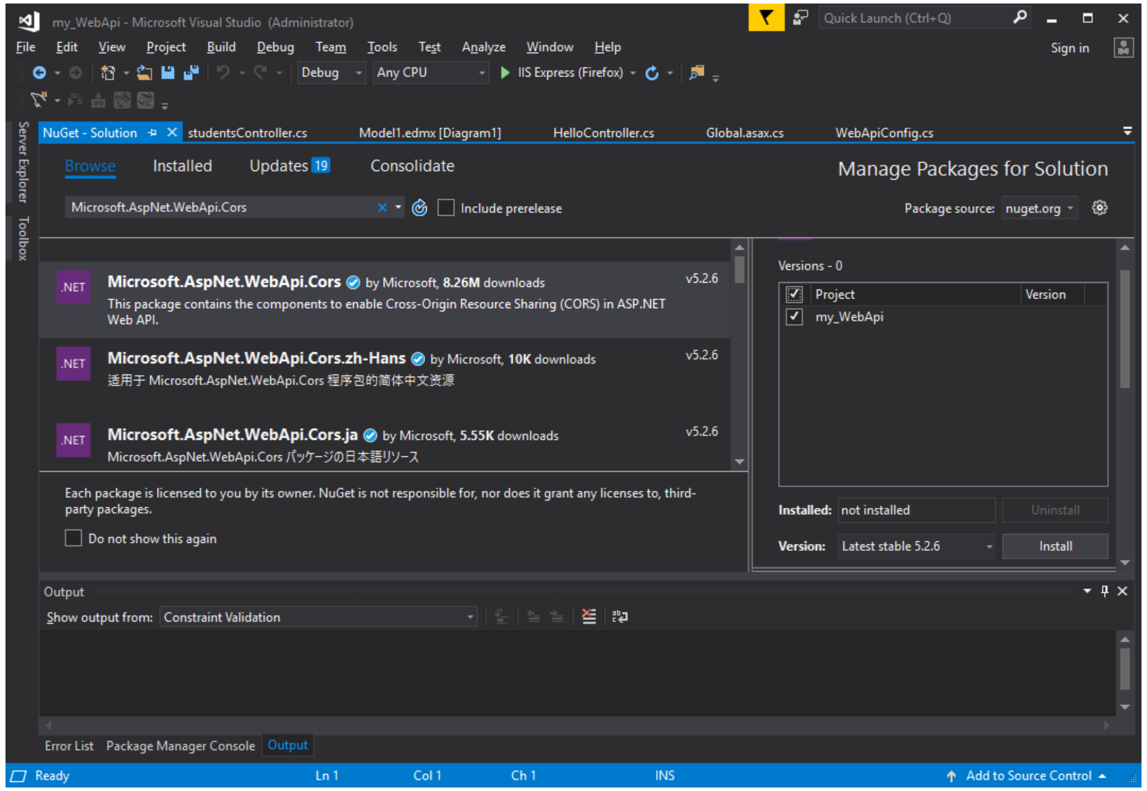
- http://example.net - Different domain
- http://example.com:9000/foo.html - Different port
- https://example.com/foo.html - Different scheme
- http://www.example.com/foo.html - Different subdomain

How to Implement Cors

To use Microsoft CORS package, you need to install from NuGet package.

Go to Tools Menu-> NuGet Package Manager -> Manage NuGet Packages for Solution

Install Package Microsoft.AspNet.WebApi.Cors



Microsoft.AspNet.WebApi.Cors

Microsoft.AspNet.WebApi.Cors by Microsoft, 8.26M downloads v5.2.6
This package contains the components to enable Cross-Origin Resource Sharing (CORS) in ASP.NET Web API.

Microsoft.AspNet.WebApi.Cors.zh-Hans by Microsoft, 10K downloads v5.2.6
适用于 Microsoft.AspNet.WebApi.Cors 程序包的简体中文资源

Microsoft.AspNet.WebApi.Cors.ja by Microsoft, 5.55K downloads v5.2.6
Microsoft.AspNet.WebApi.Cors パッケージの日本語リソース

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Installed: not installed
Version: Latest stable 5.2.6

Show output from: Constraint Validation

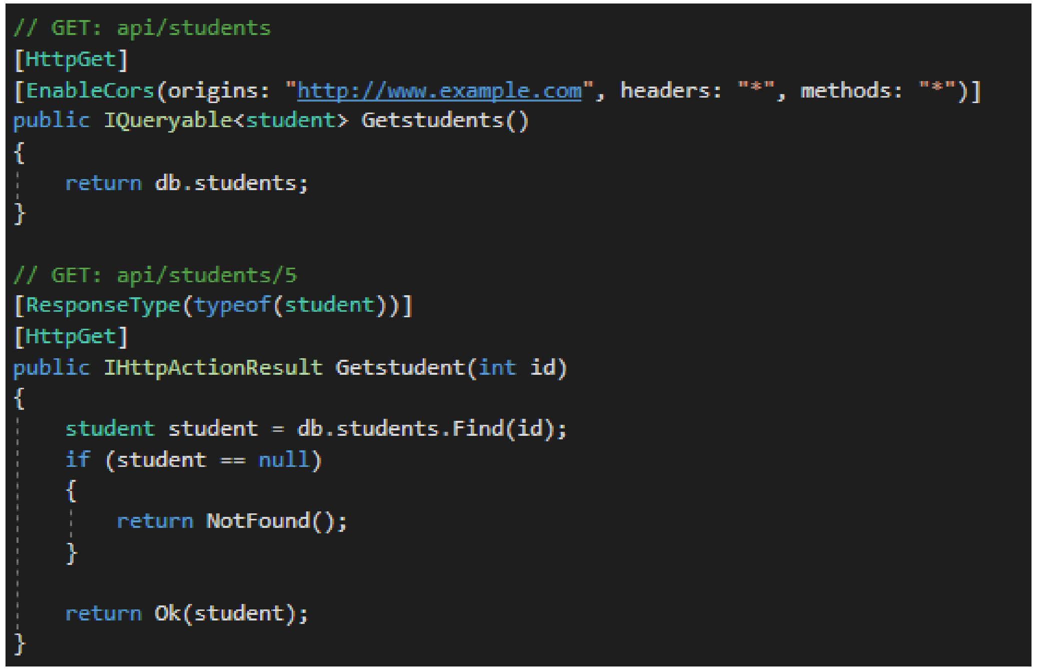
How to Implement Cors

You can configure CORS support for Web API at three levels.

- Action level
- Global level
- Controller level

Per Action

To enable CORS for a single action, set the [EnableCors] attribute on the action method. The following example enables CORS for the Getstudents method only.

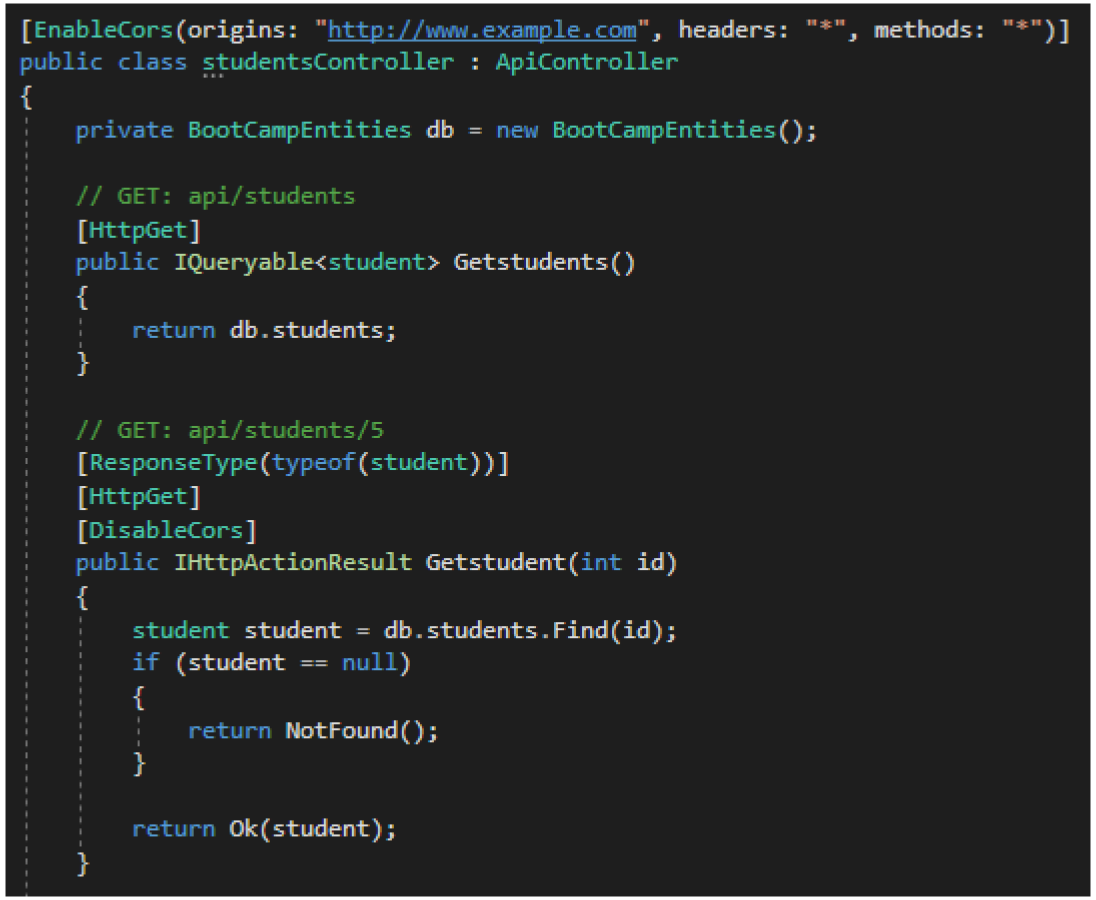


```
// GET: api/students
[HttpGet]
[EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
public IQueryable<student> Getstudents()
{
    return db.students;
}

// GET: api/students/5
[ResponseType(typeof(student))]
[HttpGet]
public IHttpActionResult Getstudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        return NotFound();
    }

    return Ok(student);
}
```

How to Implement Cors

Per Controller

If you set [EnableCors] on the controller class, it applies to all the actions on the controller. To disable CORS for an action, add the [DisableCors] attribute to the action. The following example disables CORS Getstudent method.

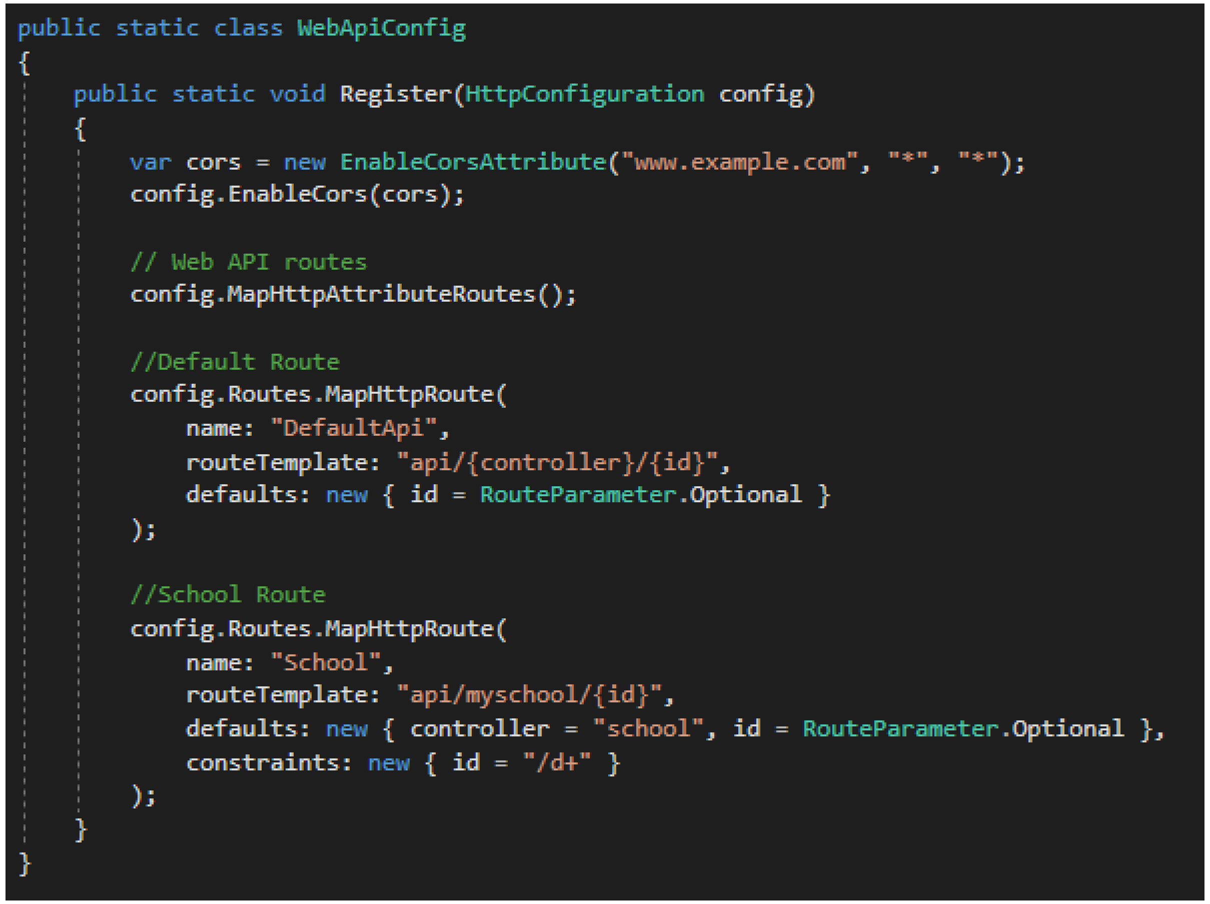


```
[EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
public class studentsController : ApiController
{
    private BootCampEntities db = new BootCampEntities();

    // GET: api/students
    [HttpGet]
    public IQueryable<student> Getstudents()
    {
        return db.students;
    }

    // GET: api/students/5
    [ResponseType(typeof(student))]
    [HttpGet]
    [DisableCors]
    public IHttpActionResult Getstudent(int id)
    {
        student student = db.students.Find(id);
        if (student == null)
        {
            return NotFound();
        }

        return Ok(student);
    }
```

How to Implement Cors

Globally

To enable CORS for all Web API controllers in your application, pass an **EnableCorsAttribute** instance to the **EnableCors** method:

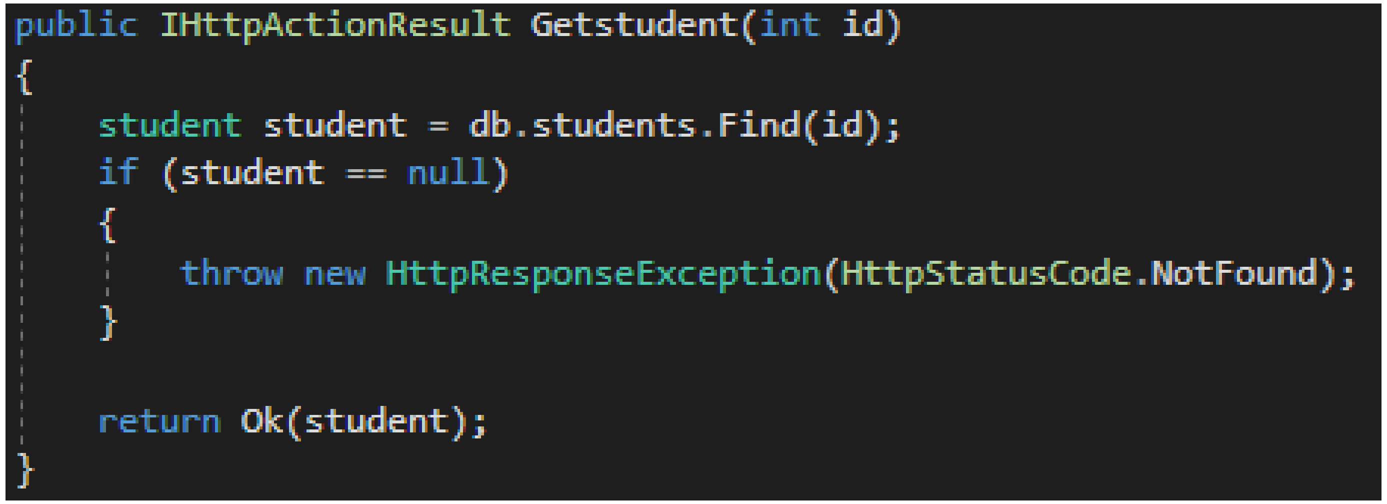


```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        var cors = new EnableCorsAttribute("www.example.com", "*", "*");
        config.EnableCors(cors);

        // Web API routes
        config.MapHttpAttributeRoutes();

        //Default Route
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

        //School Route
        config.Routes.MapHttpRoute(
            name: "School",
            routeTemplate: "api/myschool/{id}",
            defaults: new { controller = "school", id = RouteParameter.Optional },
            constraints: new { id = "/d+" }
        );
    }
}
```

Error Handling

What happens if a Web API controller throws an uncaught exception? By default, most exceptions are translated into an HTTP response with status code 500, Internal Server Error.

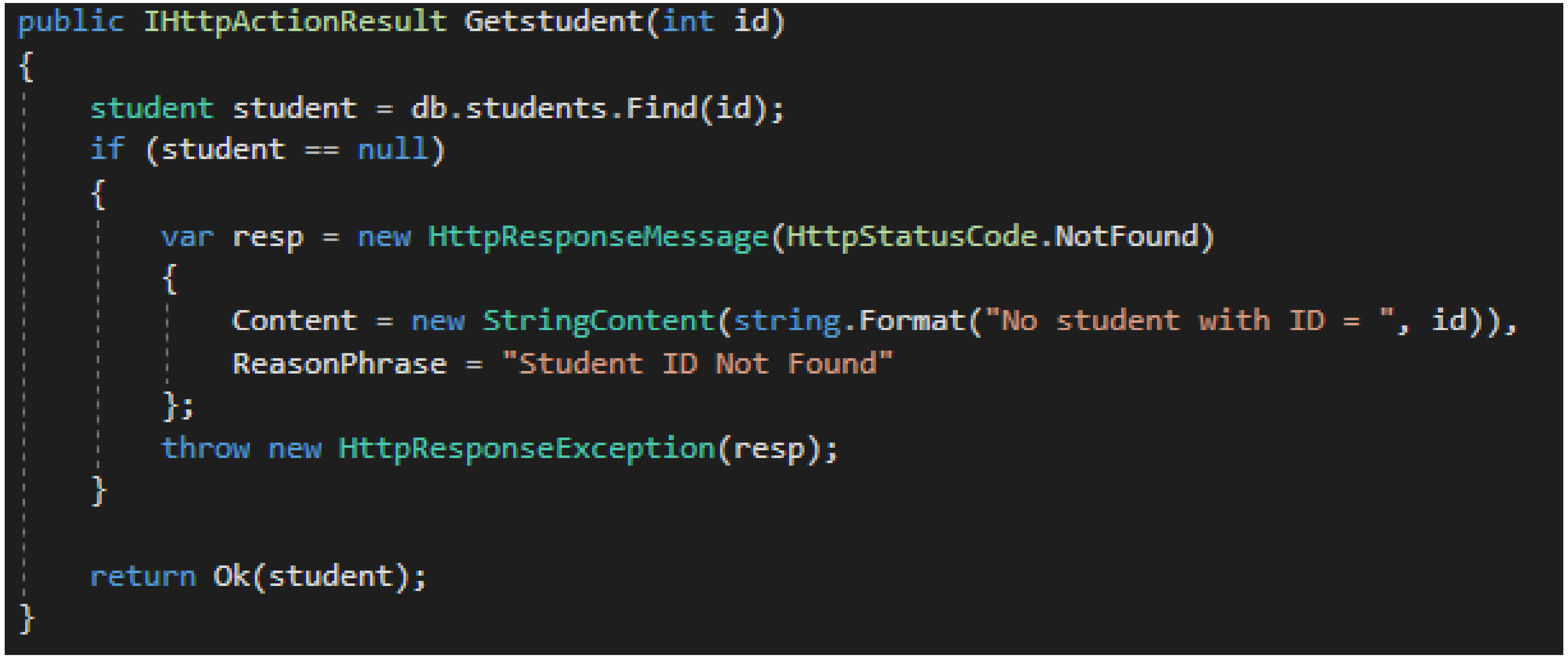
The **HttpResponseException** type is a special case. This exception returns any HTTP status code that you specify in the exception constructor. For example, the following method returns 404, Not Found, if the *id* parameter is not valid.



```
public IHttpActionResult Getstudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
    }

    return Ok(student);
}
```

Error Handling

For more control over the response, you can also construct the entire response message and include it with the **HttpResponseException:**



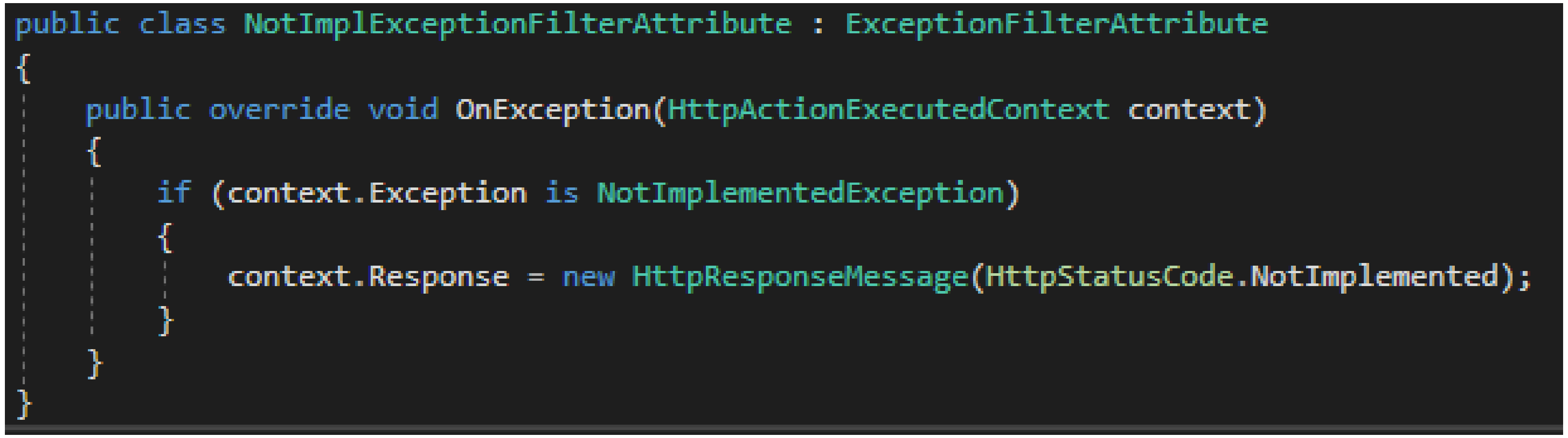
```
public IHttpActionResult Getstudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        var resp = new HttpResponseMessage(HttpStatusCode.NotFound)
        {
            Content = new StringContent(string.Format("No student with ID = ", id)),
            ReasonPhrase = "Student ID Not Found"
        };
        throw new HttpResponseException(resp);
    }

    return Ok(student);
}
```

Error Handling

Exception Filters

You can customize how Web API handles exceptions by writing an *exception filter*. An exception filter is executed when a controller method throws any unhandled exception that is *not* an **HttpResponseException** exception. The **HttpResponseException** type is a special case, because it is designed specifically for returning an HTTP response.

Here is a filter that converts **NotImplementedException** exceptions into HTTP status code 501, Not Implemented:



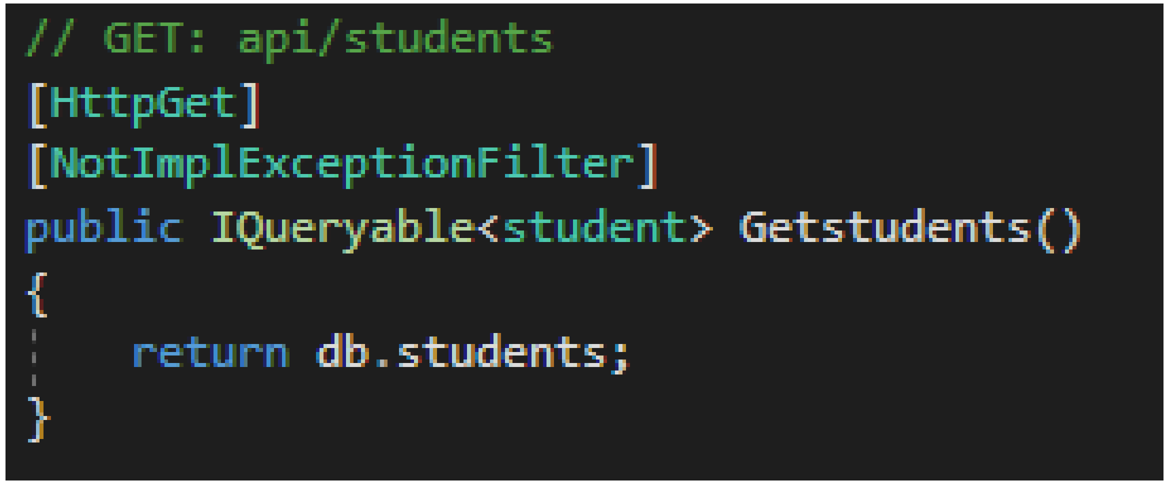
```
public class NotImplExceptionFilterAttribute : ExceptionFilterAttribute
{
    public override void OnException(HttpActionExecutedContext context)
    {
        if (context.Exception is NotImplementedException)
        {
            context.Response = new HttpResponseMessage(HttpStatusCode.NotImplemented);
        }
    }
}
```

Error Handling

Registering Exception Filters
There are several ways to register a Web API exception filter:

- By action
- By controller
- Globally

By Action
To apply the filter to a specific action, add the filter as an attribute to the action:

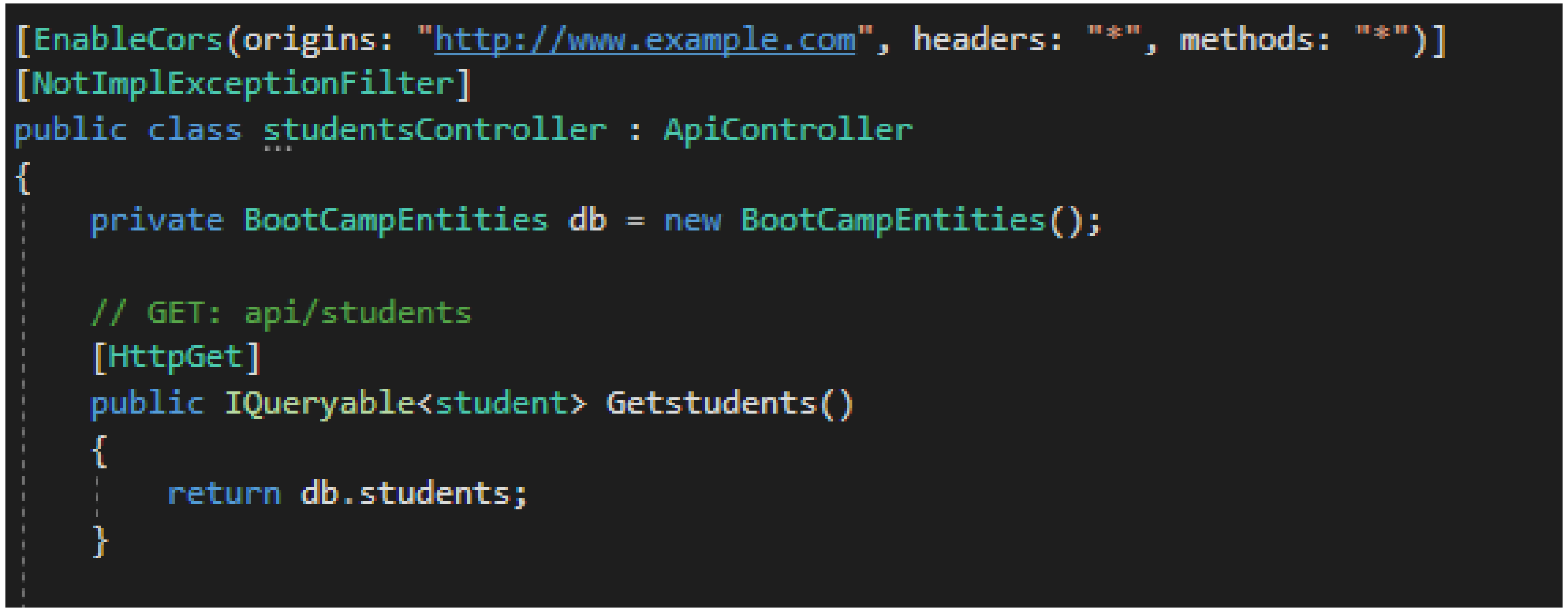


```
// GET: api/students
[HttpGet]
[NotImplExceptionFilter]
public IQueryable<student> Getstudents()
{
    return db.students;
}
```

Error Handling

By Controller

To apply the filter to all of the actions on a controller, add the filter as an attribute to the controller class:

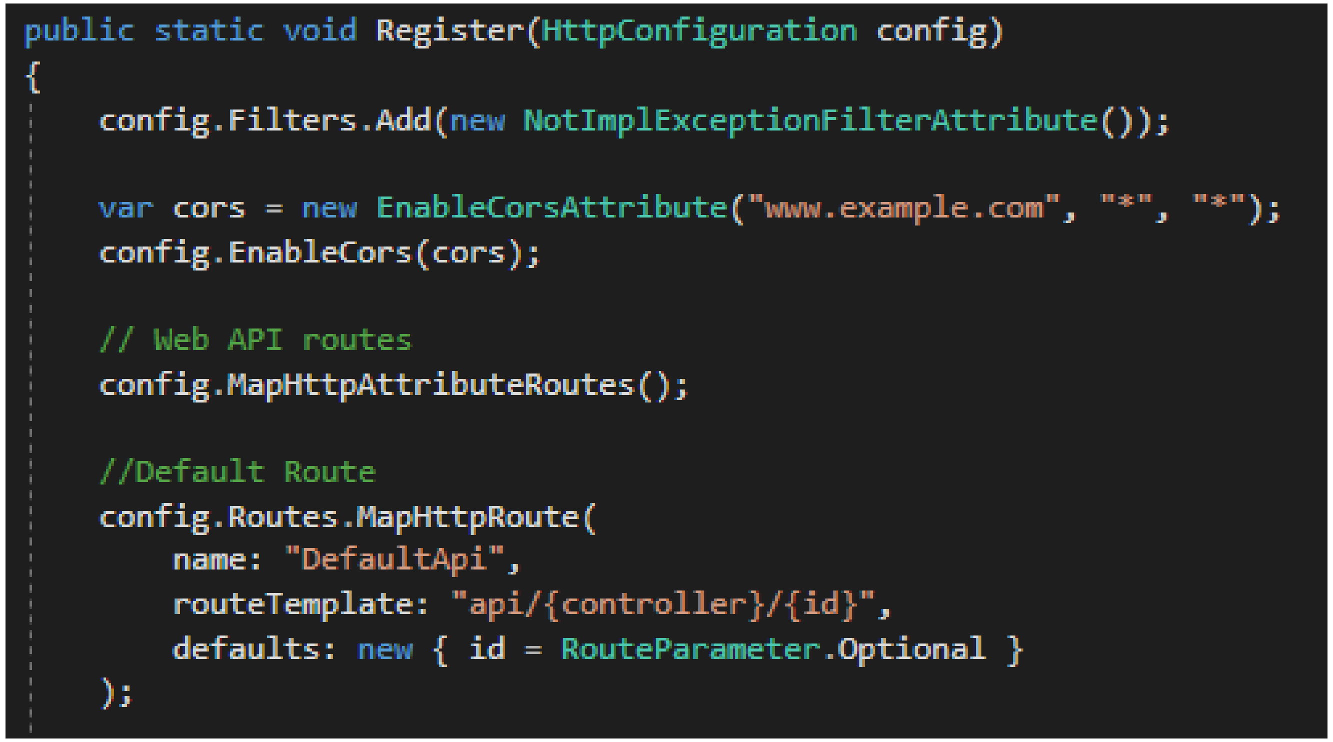


```
[EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
[NotImplExceptionFilter]
public class studentsController : ApiController
{
    private BootCampEntities db = new BootCampEntities();

    // GET: api/students
    [HttpGet]
    public IQueryable<student> Getstudents()
    {
        return db.students;
    }
```

Error Handling

Globally

To apply the filter globally to all Web API controllers, add an instance of the filter to the **GlobalConfiguration.Configuration.Filters** collection. Exeption filters in this collection apply to any Web API controller action.



```
public static void Register(HttpConfiguration config)
{
    config.Filters.Add(new NotImplExceptionFilterAttribute());

    var cors = new EnableCorsAttribute("www.example.com", "*", "*");
    config.EnableCors(cors);

    // Web API routes
    config.MapHttpAttributeRoutes();

    //Default Route
    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
```

Security

Authentication

Globally

An authentication filter is a component that authenticates an HTTP request. Web API 2 and MVC 5 both support authentication filters, but they differ slightly, mostly in the naming conventions for the filter interface

Authentication filters let you set an authentication scheme for individual controllers or actions. That way, your app can support different authentication mechanisms for different HTTP resources.

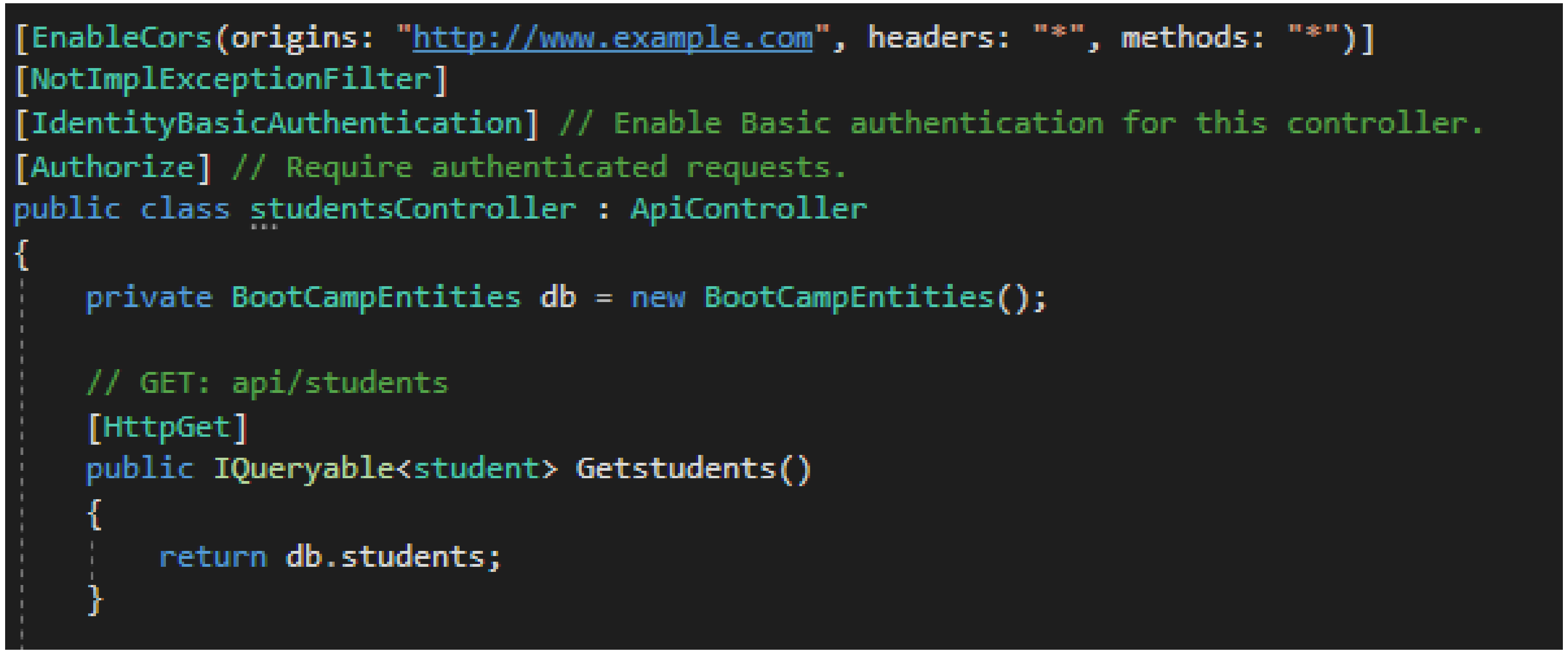
Like other filters, authentication filters can be applied to:

- Controller
- Action
- Globally

Authentication

By Controller

To apply an authentication filter to a controller, decorate the controller class with the filter attribute. The following code sets the [IdentityBasicAuthentication] filter on a controller class, which enables Basic Authentication for all of the controller's actions.

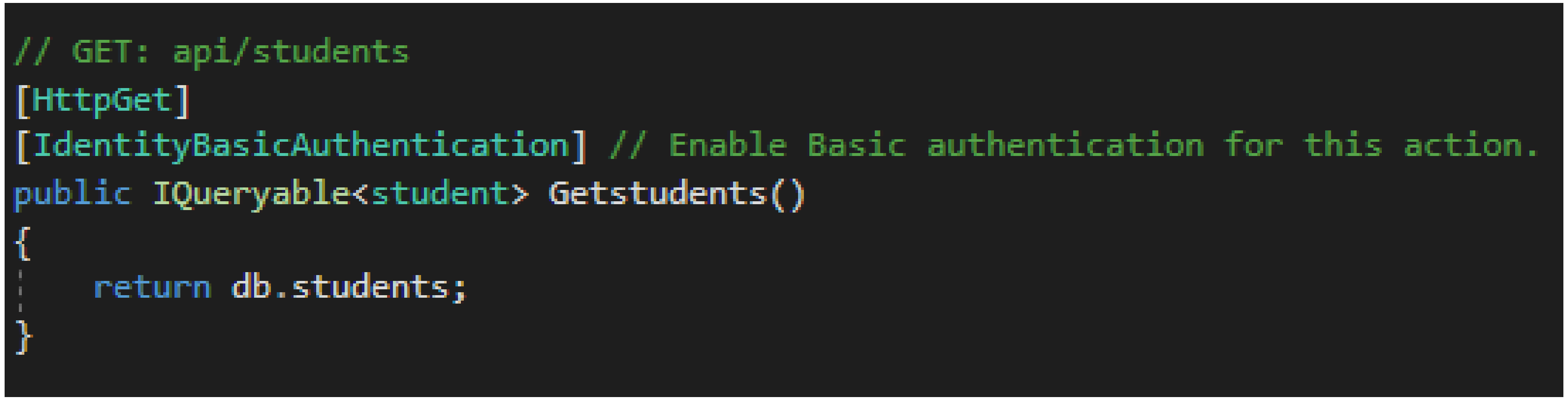


```
[EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
[NotImplExceptionFilter]
[IdentityBasicAuthentication] // Enable Basic authentication for this controller.
[Authorize] // Require authenticated requests.
public class studentsController : ApiController
{
    private BootCampEntities db = new BootCampEntities();

    // GET: api/students
    [HttpGet]
    public IQueryable<student> Getstudents()
    {
        return db.students;
    }
```

Authentication

By Action

To apply the filter to one action, decorate the action with the filter. The following code sets the [IdentityBasicAuthentication] filter on the controller's Post method.

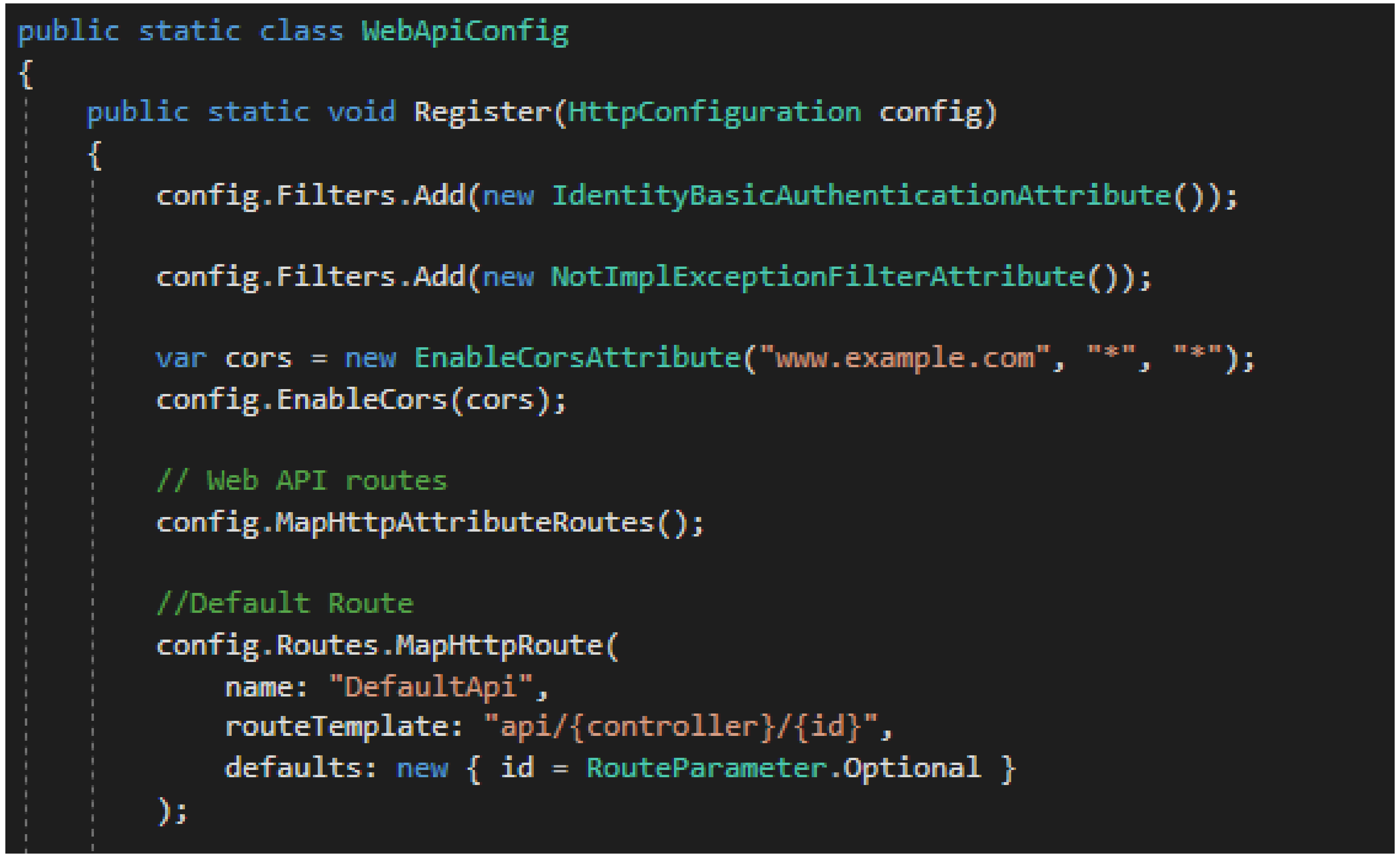


```
// GET: api/students
[HttpGet]
[IdentityBasicAuthentication] // Enable Basic authentication for this action.
public IQueryable<student> Getstudents()
{
    return db.students;
}
```

Authentication

Globally

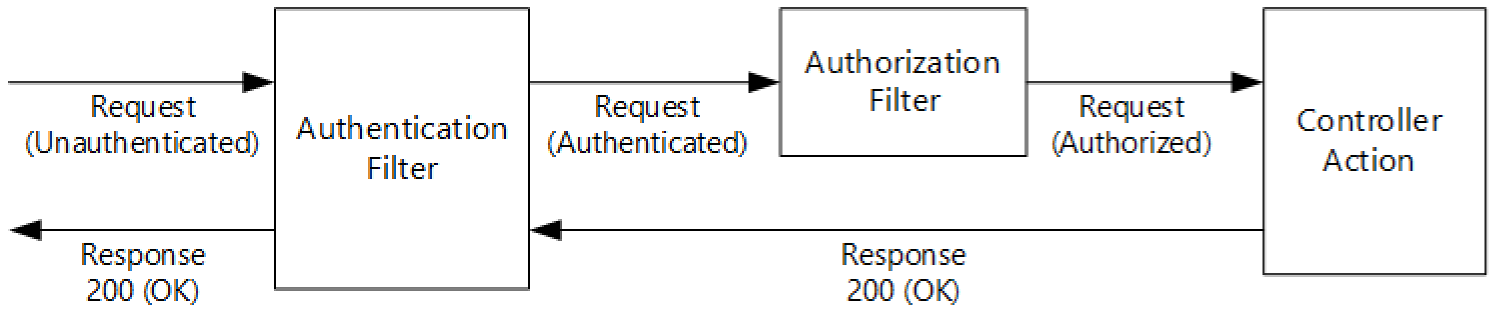
To apply the filter to all Web API controllers, add it to **GlobalConfiguration.Filters**.



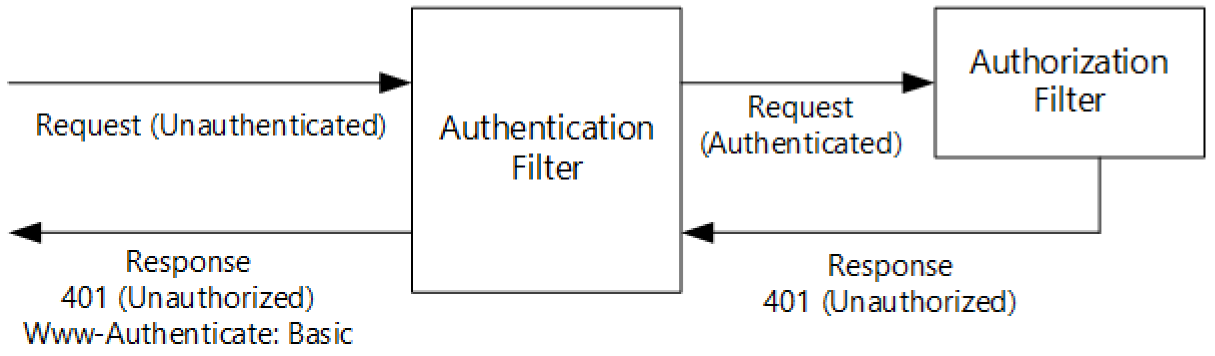
```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.Filters.Add(new IdentityBasicAuthenticationAttribute());

        config.Filters.Add(new NotImplExceptionFilterAttribute());

        var cors = new EnableCorsAttribute("www.example.com", "*", "*");
        config.EnableCors(cors);

        // Web API routes
        config.MapHttpAttributeRoutes();

        //Default Route
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
```

Authentication

The authentication filter successfully authenticates the request, an authorization filter authorizes the request, and the controller action returns 200 (OK).



the authentication filter authenticates the request, but the authorization filter returns 401 (Unauthorized). In this case, the controller action is not invoked. The authentication filter adds a Www-Authenticate header to the response.


Request (Unauthenticated)
Authentication Filter
Request (Authenticated)
Authorization Filter
Response 401 (Unauthorized) Www-Authenticate: Basic
Response 401 (Unauthorized)

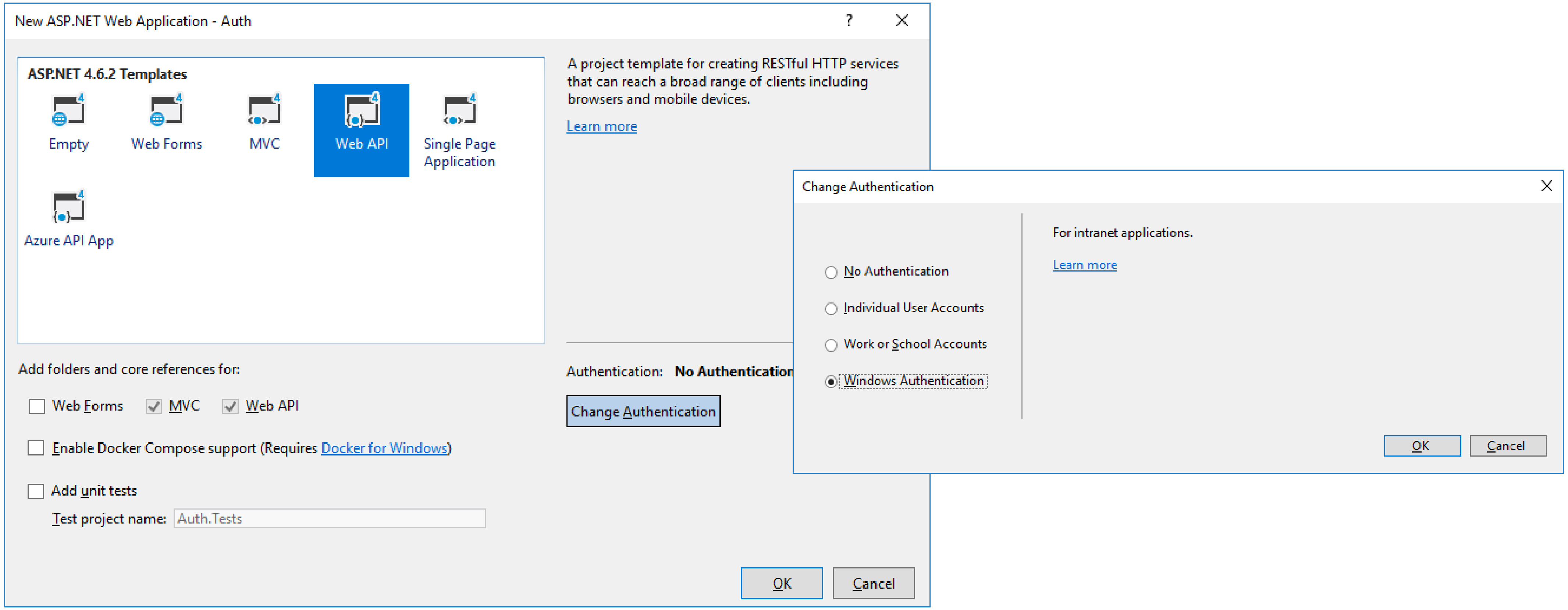
Integrated Windows Authentication

Integrated Windows authentication enables users to log in with their Windows credentials, using Kerberos or NTLM. The client sends credentials in the Authorization header. Windows authentication is best suited for an intranet environment.

Advantages	Disadvantages
• Built into IIS. - Does not send the user credentials in the request. • If the client computer belongs to the domain (for example, intranet application), the user does not need to enter credentials.	• Not recommended for Internet applications. • Requires Kerberos or NTLM support in the client. • Client must be in the Active Directory domain.

Integrated Windows Authentication

To create an application that uses Integrated Windows authentication, click on "Change Authentication" and select Windows Authentication.


New ASP.NET Web Application - Auth

ASP.NET 4.6.2 Templates

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Learn more

Add folders and core references for:

☐ Web Forms ☑ MVC ☑ Web API

☐ Enable Docker Compose support (Requires Docker for Windows)

☐ Add unit tests

Test project name: Auth.Tests

Authentication: **No Authentication**

Change Authentication

○ No Authentication
○ Individual User Accounts
○ Work or School Accounts
[✓ Windows Authentication]

For intranet applications.

Learn more

Forms Authentication

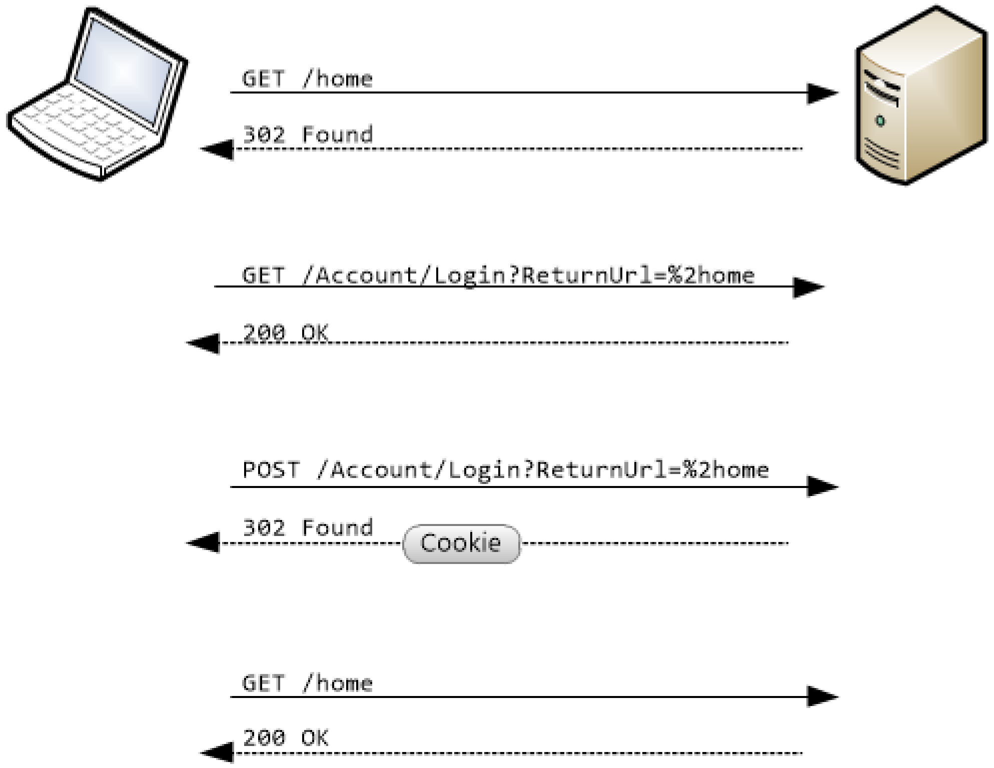
Forms authentication uses an HTML form to send the user's credentials to the server. It is not an Internet standard. Forms authentication is only appropriate for web APIs that are called from a web application, so that the user can interact with the HTML form.

Advantages	Disadvantages
• Easy to implement: Built into ASP.NET • Uses ASP.NET membership provider, which makes it easy to manage user accounts.	• Not a standard HTTP authentication mechanism; uses HTTP cookies instead of the standard Authorization header. • Requires a browser client. • Credentials are sent as plaintext • Vulnerable to cross-site request forgery (CSRF); requires anti-CSRF measures. • Difficult to use from nonbrowser clients. Login requires a browser. • User credentials are sent in the request. - Some users disable cookies.

Forms Authentication

Forms authentication in ASP.NET works like this:

1. The client requests a resource that requires authentication.
2. If the user is not authenticated, the server returns HTTP 302 (Found) and redirects to a login page.
3. The user enters credentials and submits the form.
4. The server returns another HTTP 302 that redirects back to the original URI. This response includes an authentication cookie.
5. The client requests the resource again. The request includes the authentication cookie, so the server grants the request.

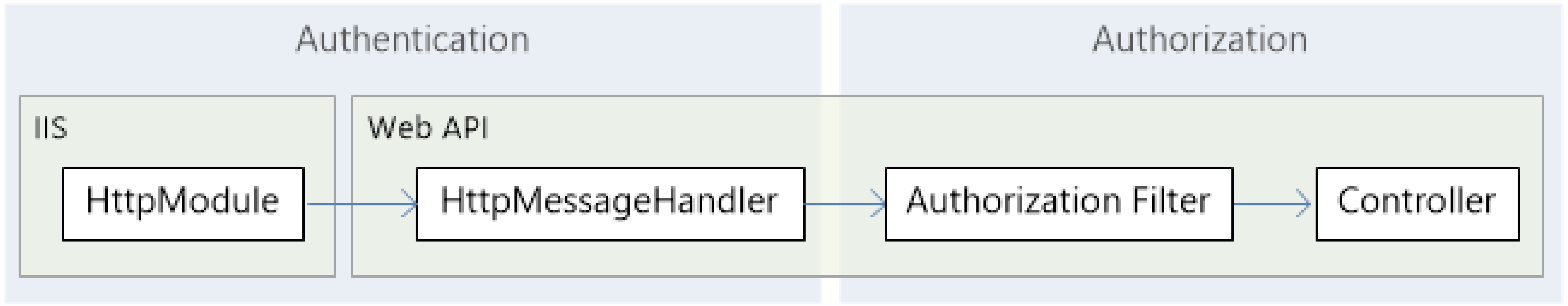

GET /home
302 Found
GET /Account/Login?ReturnUrl=%2home
200 OK
POST /Account/Login?ReturnUrl=%2home
302 Found
Cookie
GET /home
200 OK

Authorization

Authorization happens later in the pipeline, closer to the controller. That lets you make more granular choices when you grant access to resources.

- *Authorization filters* run before the controller action. If the request is not authorized, the filter returns an error response, and the action is not invoked.

- Within a controller action, you can get the current principal from the **ApiController.User**property. For example, you might filter a list of resources based on the user name, returning only those resources that belong to that user.



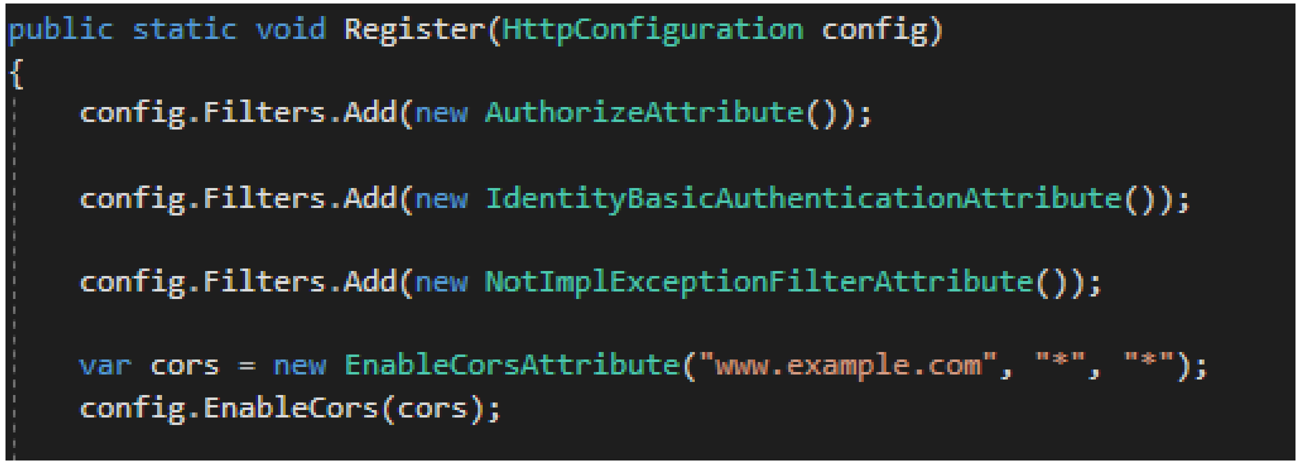
Authorization

Using the [Authorize] Attribute

Web API provides a built-in authorization filter, AuthorizeAttribute. This filter checks whether the user is authenticated. If not, it returns HTTP status code 401 (Unauthorized), without invoking the action.
You can apply the filter:

- Globally
- Controller level
- Individual actions.

Globally
To restrict access for every Web API controller, add the **AuthorizeAttribute** filter to the global filter list:

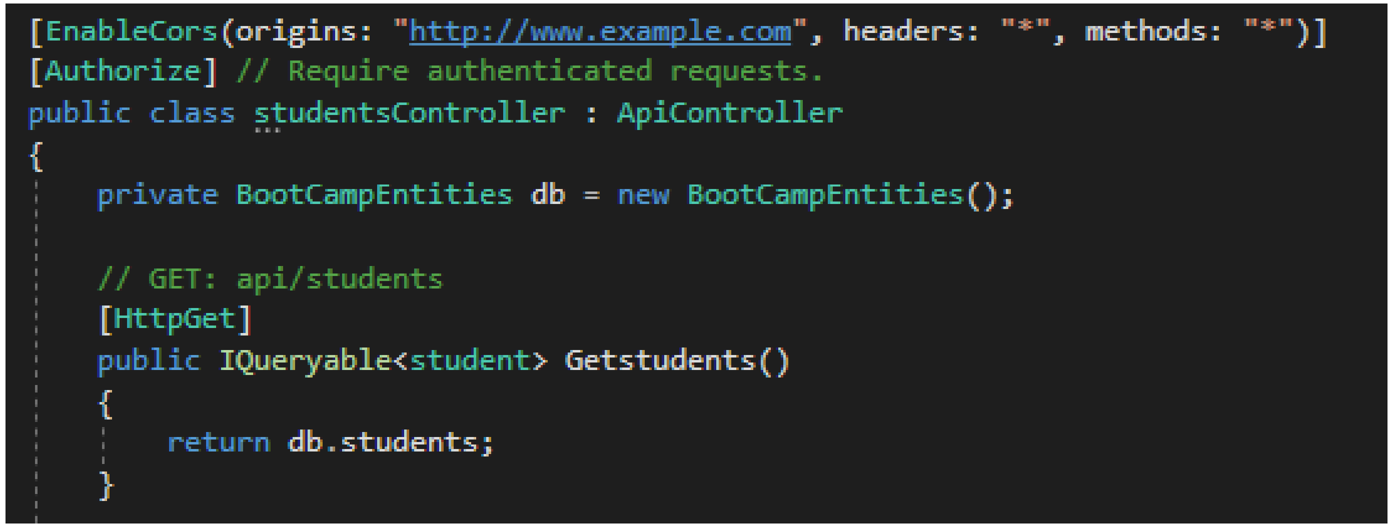


```
public static void Register(HttpConfiguration config)
{
    config.Filters.Add(new AuthorizeAttribute());

    config.Filters.Add(new IdentityBasicAuthenticationAttribute());

    config.Filters.Add(new NotImplExceptionFilterAttribute());

    var cors = new EnableCorsAttribute("www.example.com", "*", "*");
    config.EnableCors(cors);
```

Authorization

Controller

To restrict access for a specific controller, add the filter as an attribute to the controller:

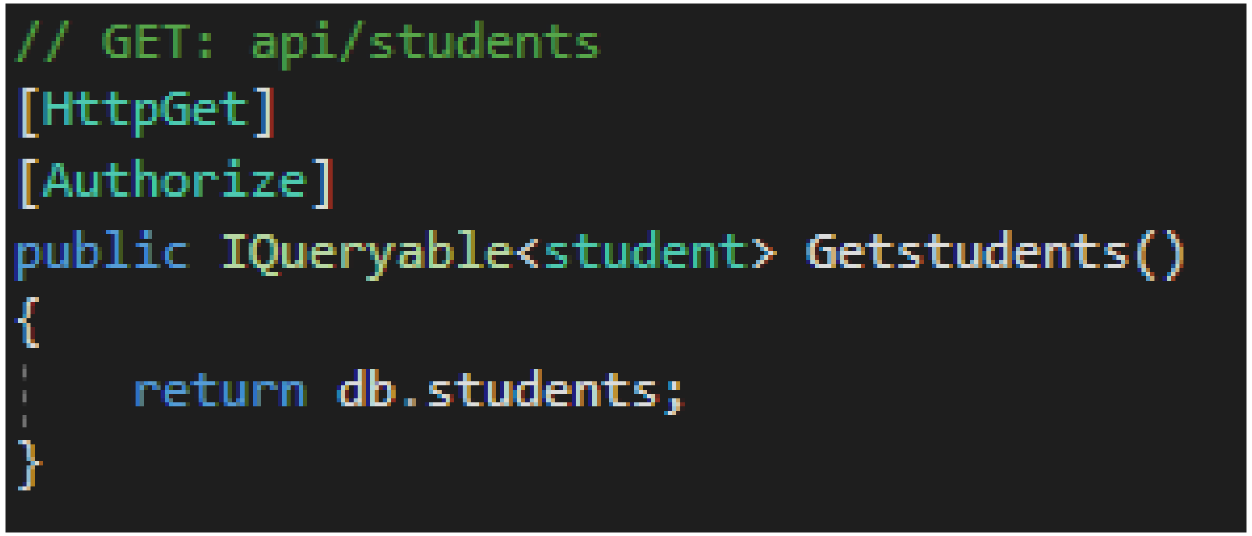


```
[EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
[Authorize] // Require authenticated requests.
public class studentsController : ApiController
{
    private BootCampEntities db = new BootCampEntities();

    // GET: api/students
    [HttpGet]
    public IQueryable<student> Getstudents()
    {
        return db.students;
    }
```

Authorization

Action

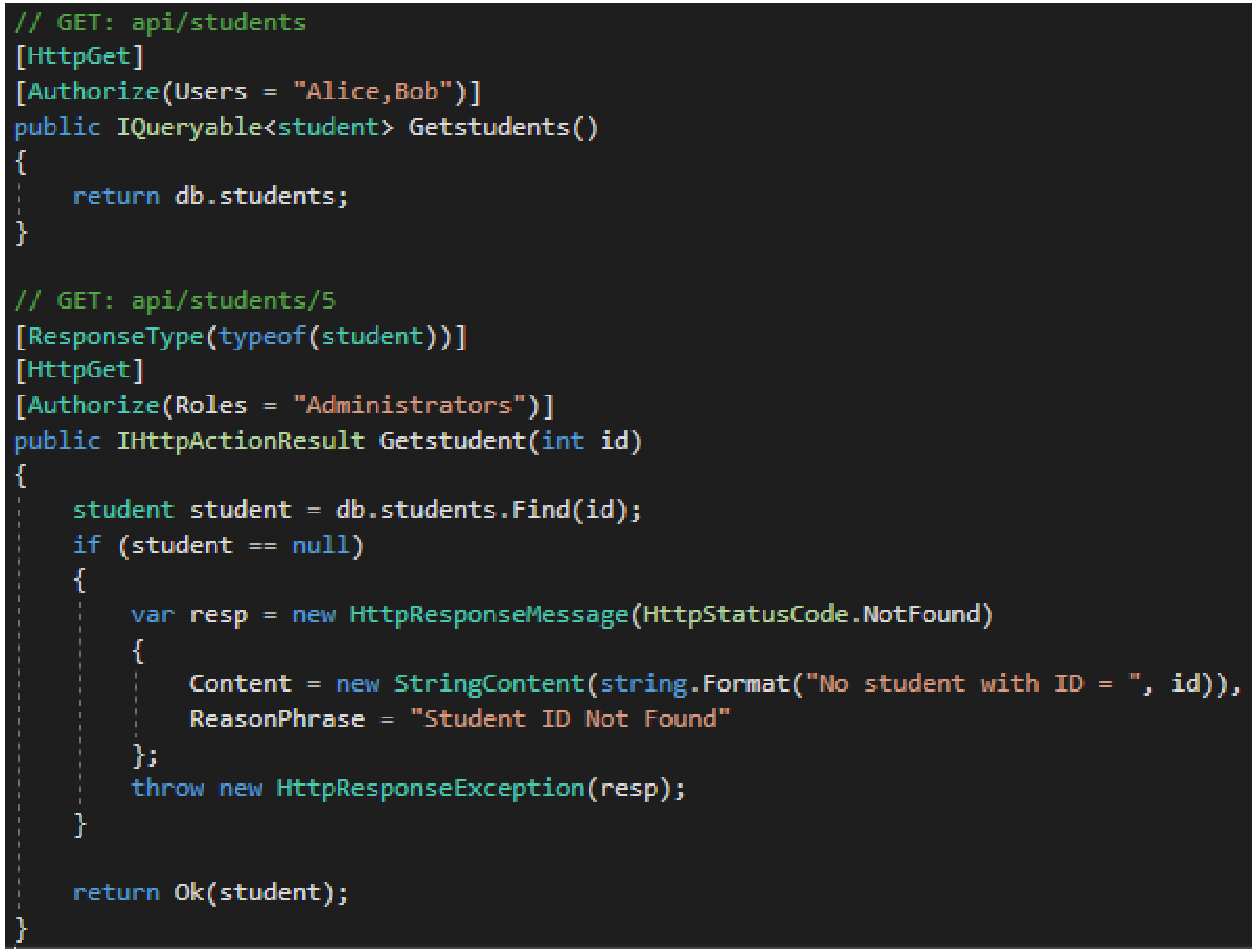
To restrict access for specific actions, add the attribute to the action method:



```
// GET: api/students
[HttpGet]
[Authorize]
public IQueryable<student> Getstudents()
{
    return db.students;
}
```

Authorization

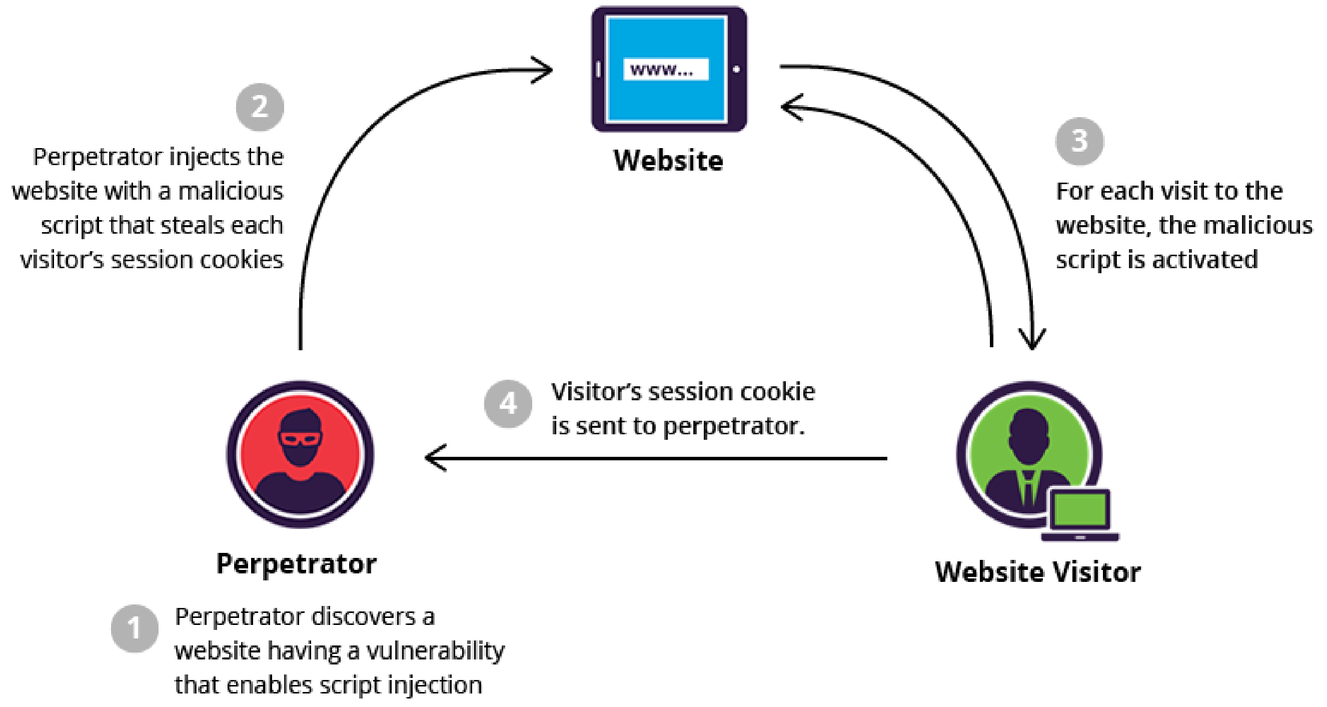
The filter allows any authenticated user to access the restricted methods; only anonymous users are kept out. You can also limit access to specific users or to users in specific roles:



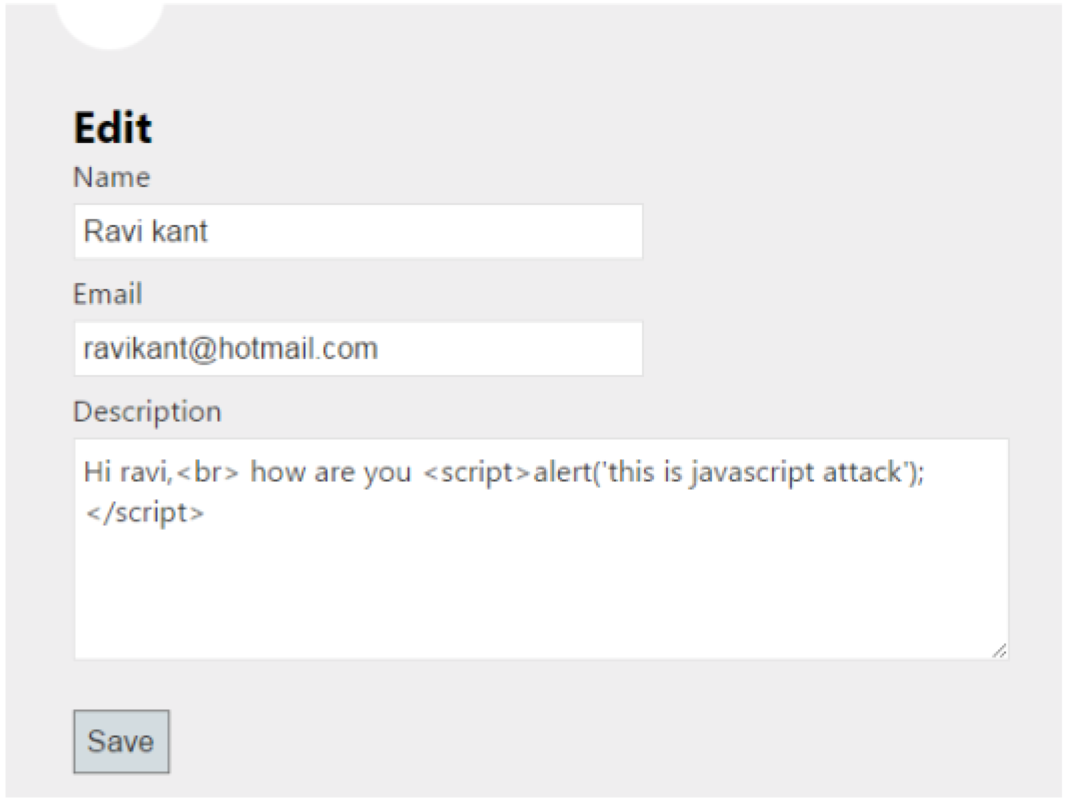
```
// GET: api/students
[HttpGet]
[Authorize(Users = "Alice,Bob")]
public IQueryable<student> Getstudents()
{
    return db.students;
}

// GET: api/students/5
[ResponseType(typeof(student))]
[HttpGet]
[Authorize(Roles = "Administrators")]
public IHttpActionResult Getstudent(int id)
{
    student student = db.students.Find(id);
    if (student == null)
    {
        var resp = new HttpResponseMessage(HttpStatusCode.NotFound)
        {
            Content = new StringContent(string.Format("No student with ID = ", id)),
            ReasonPhrase = "Student ID Not Found"
        };
        throw new HttpResponseException(resp);
    }

    return Ok(student);
}
```

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. The consequences of XSS may range from petty nuisance like displaying an alert box to a significant security risk like stealing session cookies.


www...
Website
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Cross-Site Scripting (XSS)


Edit

Name

Ravi kant

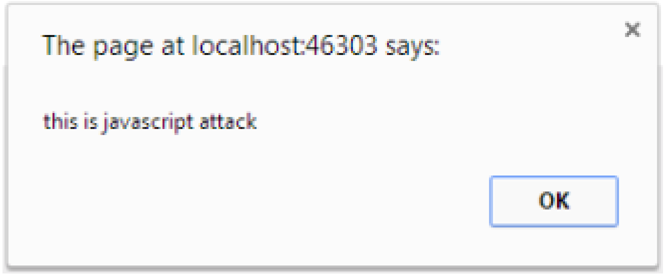
Email

ravikant@hotmail.com

Description

```
Hi ravi,<br> how are you <script>alert('this is javascript attack');
</script>
```

Save

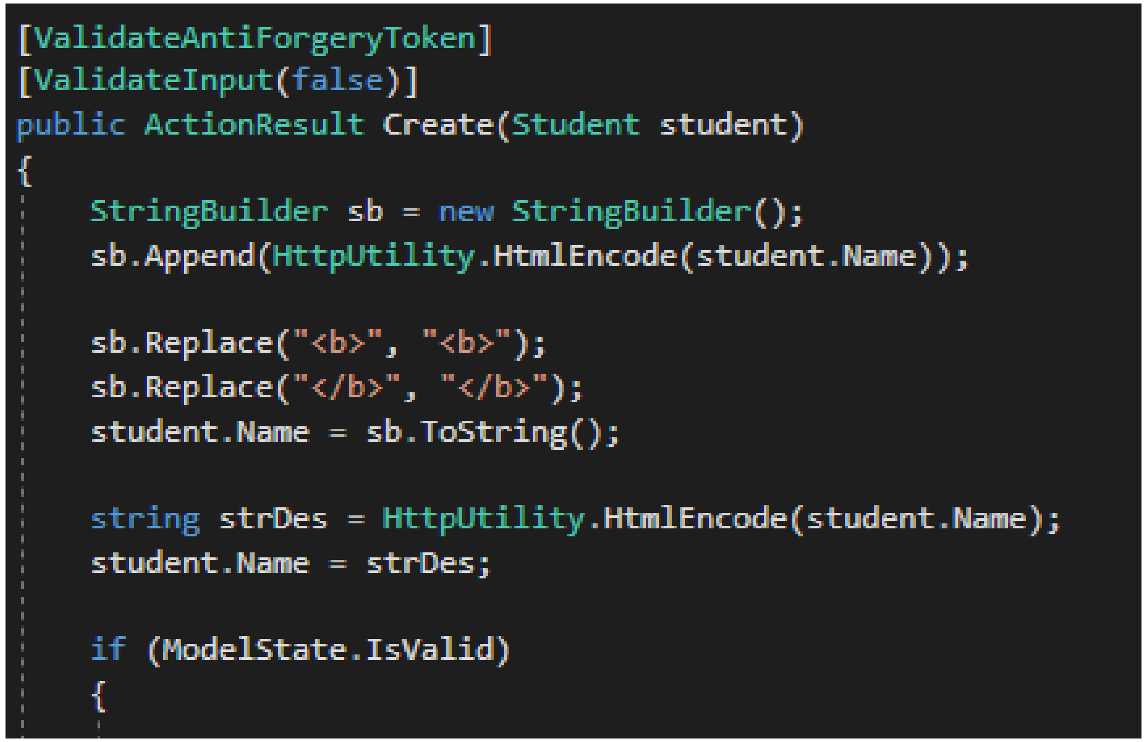

The page at localhost:46303 says:

this is javascript attack

OK

Cross-Site Scripting (XSS)

So how to prevent this scripting attack?
So for those kinds of scenarios where we want HTML to be posted we can decorate the action with "ValidateInput" set to false. This by passes the HTML and Script tag checks for that action.

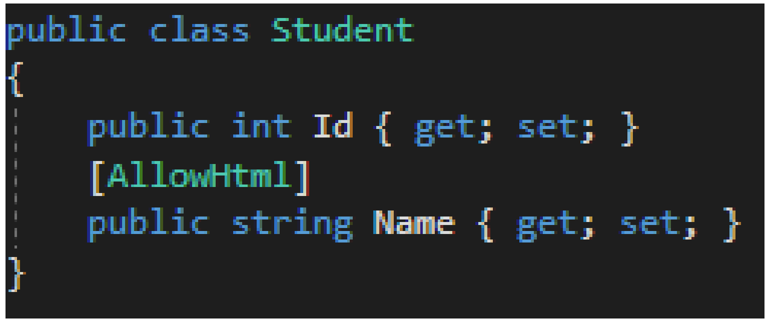


```
[ValidateAntiForgeryToken]
[ValidateInput(false)]
public ActionResult Create(Student student)
{
    StringBuilder sb = new StringBuilder();
    sb.Append(HttpUtility.HtmlEncode(student.Name));

    sb.Replace("<b>", "<b>");
    sb.Replace("</b>", "</b>");
    student.Name = sb.ToString();

    string strDes = HttpUtility.HtmlEncode(student.Name);
    student.Name = strDes;

    if (ModelState.IsValid)
    {
```

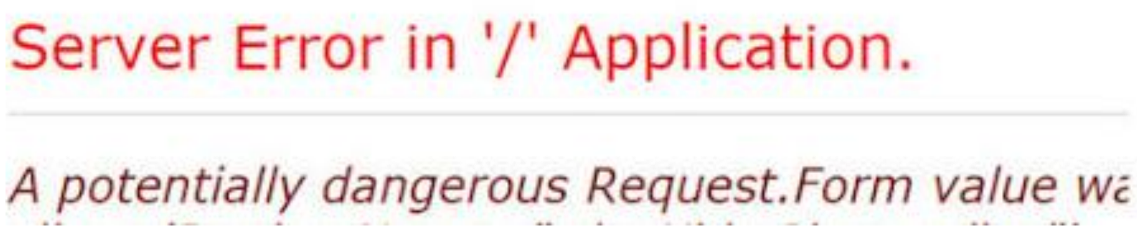
Cross-Site Scripting (XSS)

But because we have now decorated validate false at the action level , you can also write HTML in product name field as well. We would love to have more finer control on the field level rather than making the complete action naked.

That’s where “AllowHTML” comes to help. You can see in the below code we have just decorated the “ProductDescription” property .



```
public class Student
{
    public int Id { get; set; }
    [AllowHtml]
    public string Name { get; set; }
}
```



Server Error in '/' Application.

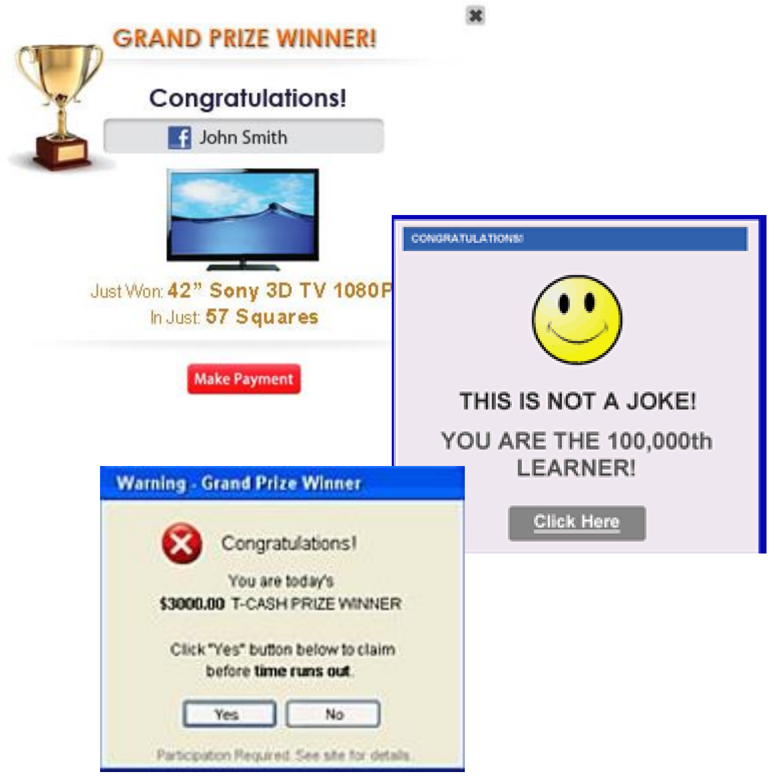
A potentially dangerous Request.Form value wa

Cross site request forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack where a malicious site sends a request to a vulnerable site where the user is currently logged in.

Here is an example of a CSRF attack:

1. A user logs into www.example.com using forms authentication.
2. The server authenticates the user. The response from the server includes an authentication cookie.
3. Without logging out, the user visits a malicious web site.
4. The user clicks the submit button. The browser includes the authentication cookie with the request.
5. The request runs on the server with the user's authentication context, and can do anything that an authenticated user is allowed to do.



GRAND PRIZE WINNER!

Congratulations!

John Smith

Just Won: 42" Sony 3D TV 1080P

In Just: 57 Squares

CONGRATULATIONS!

THIS IS NOT A JOKE!

YOU ARE THE 100,000th LEARNER!

Warning - Grand Prize Winner

Congratulations!

You are today's

$3000.00 T-CASH PRIZE WINNER

Click "Yes" button below to claim before time runs out.

Participation Required. See site for details.

Cross site request forgery (CSRF)

Anti-Forgery Tokens

To help prevent CSRF attacks, ASP.NET MVC uses anti-forgery tokens, also called *request verification tokens*.

1. The client requests an HTML page that contains a form.
2. The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.
3. When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)
4. If a request does not include both tokens, the server disallows the request.

Anti-forgery tokens work because the malicious page cannot read the user's tokens, due to same-origin policies.

To prevent CSRF attacks, use anti-forgery tokens with any authentication protocol where the browser silently sends credentials after the user logs in. This includes cookie-based authentication protocols, such as forms authentication, as well as protocols such as Basic and Digest authentication.

You should require anti-forgery tokens for any nonsafe methods (POST, PUT, DELETE).

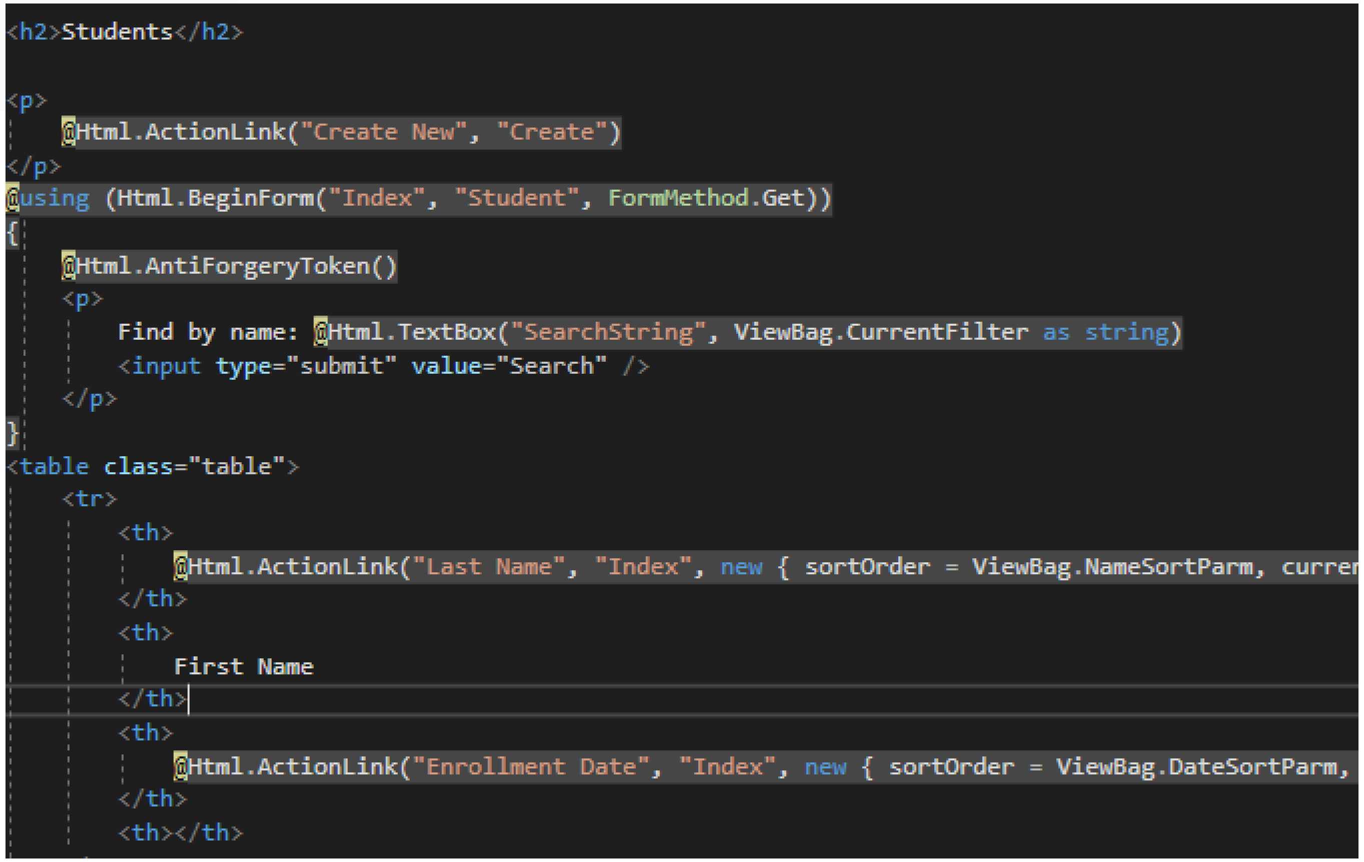
if you enable cross-domain support, such as CORS or JSONP, then even safe methods like GET are potentially vulnerable to CSRF attacks

Cross site request forgery (CSRF)

Anti-Forgery Tokens in ASP.NET MVC

To add the anti-forgery tokens to a Razor page, use the **HtmlHelper.AntiForgeryToken** helper method:

```
@using (Html.BeginForm("Manage", "Account")) {
    @Html.AntiForgeryToken()
}
```



```
<h2>Students</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
@using (Html.BeginForm("Index", "Student", FormMethod.Get))
{
    @Html.AntiForgeryToken()
    <p>
        Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
        <input type="submit" value="Search" />
    </p>
}
<table class="table">
    <tr>
        <th>
            @Html.ActionLink("Last Name", "Index", new { sortOrder = ViewBag.NameSortParm, currer
        </th>
        <th>
            First Name
        </th>
        <th>
            @Html.ActionLink("Enrollment Date", "Index", new { sortOrder = ViewBag.DateSortParm,
        </th>
        <th></th>
```

Unit Testing

What is unit Testing?

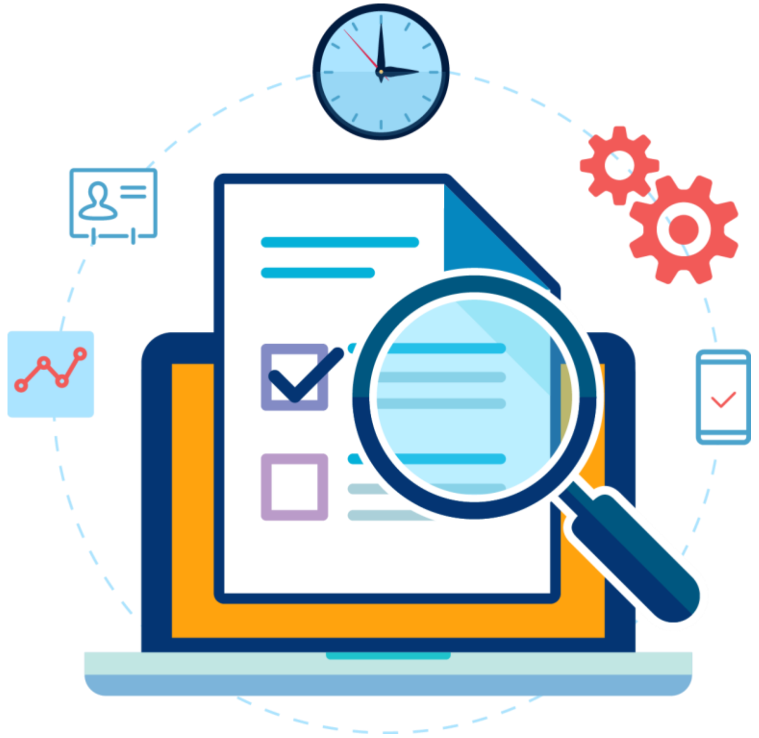
In this IT world a unit refers to simply a smallest piece of code which takes an input ,does certain operation, and gives an output.
And testing this small piece of code is called Unit Testing.

A lot of unit test frameworks are available for .Net nowadays, if we check in Visual Studio we have MSTest from Microsoft integrated in Visual Studio.

Some 3rd party frameworks are:

- NUnit
- MbUnit

Out of all these Nunit is the most-used testing Framework



Unit Testing

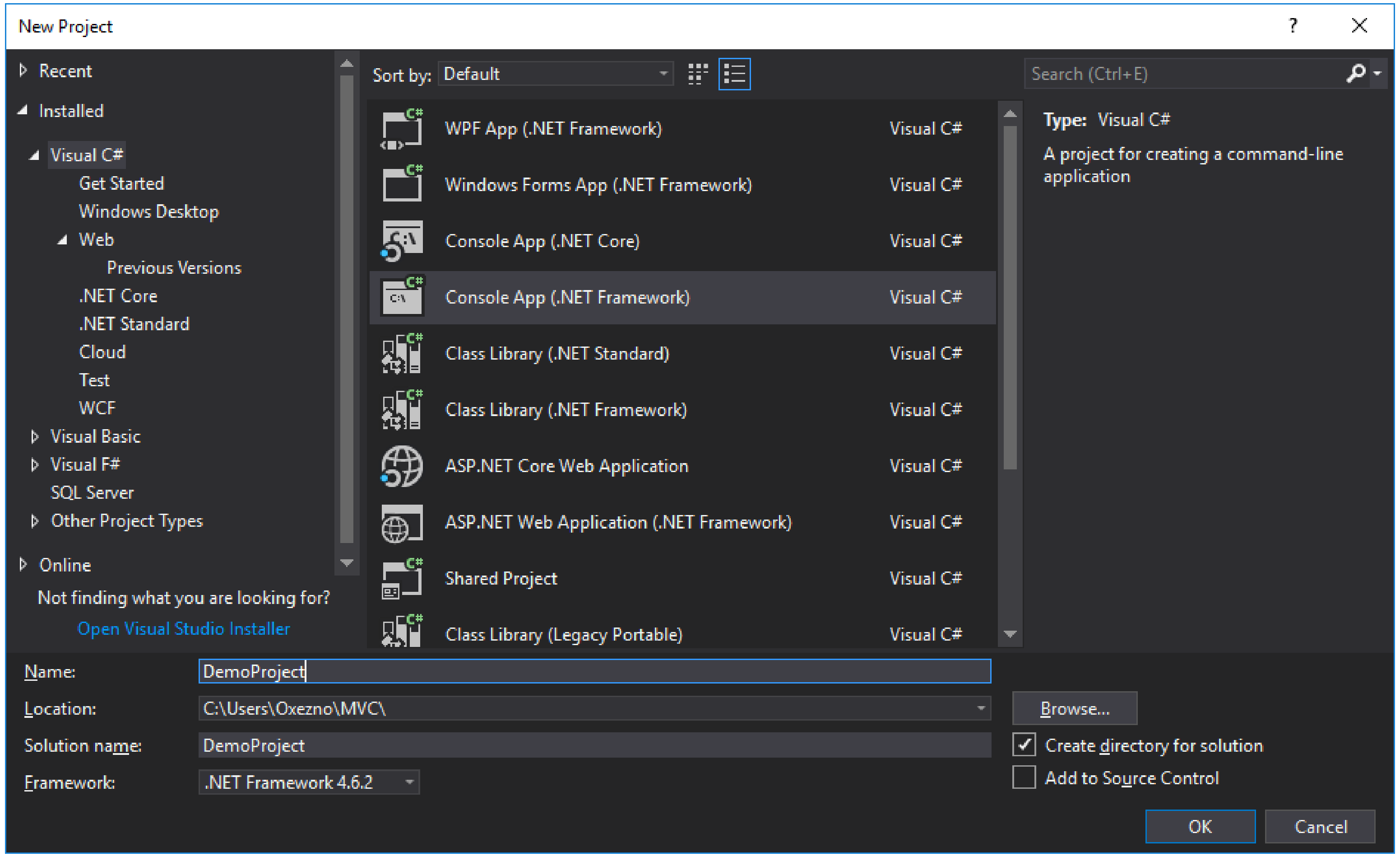
What Is NUnit?

- NUnit is a unit-testing framework for all .Net languages.
- NUnit is Open Source software and NUnit 3.0 is released under the MIT license. This framework is very easy to work with and has user friendly attributes for working.

The Nunit framework does not require any specific project type, but most of the time people will add a class library to separate their code from their unit tests. You need to reference the nunit.framework.dll yourself. But if you use nuget this will be done for you automatically while installing NUnit.

Unit Testing

To work with Nunit let's set up a simple console application.



New Project

- Recent
- Installed
  - Visual C#
    - Get Started
    - Windows Desktop
    - Web
      - Previous Versions
    - .NET Core
    - .NET Standard
    - Cloud
    - Test
    - WCF
  - Visual Basic
  - Visual F#
  - SQL Server
  - Other Project Types
- Online

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

Search (Ctrl+E)

WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Standard)	Visual C#
Class Library (.NET Framework)	Visual C#
ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#
Shared Project	Visual C#
Class Library (Legacy Portable)	Visual C#

Type: Visual C#

A project for creating a command-line application

Name: DemoProject

Location: C:\Users\Oxezno\MVC\

Solution name: DemoProject

Framework: .NET Framework 4.6.2

Browse...

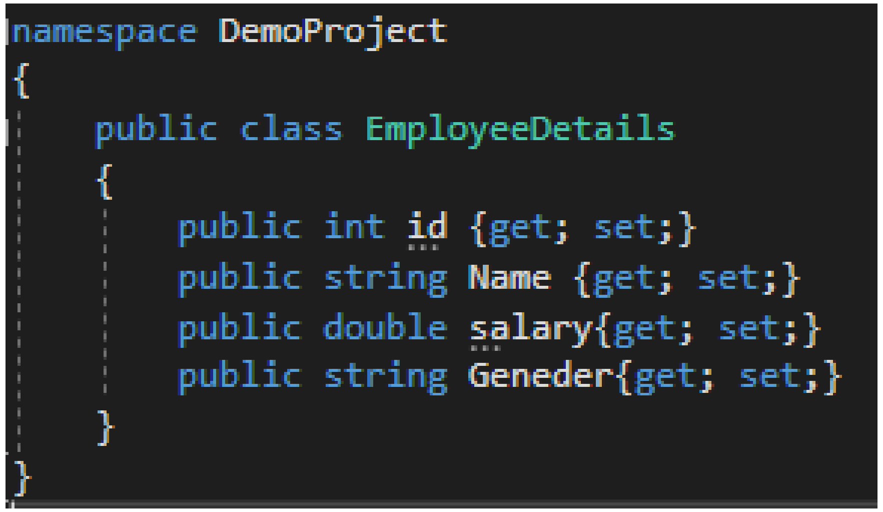
☑ Create directory for solution

☐ Add to Source Control

OK Cancel

Unit Testing

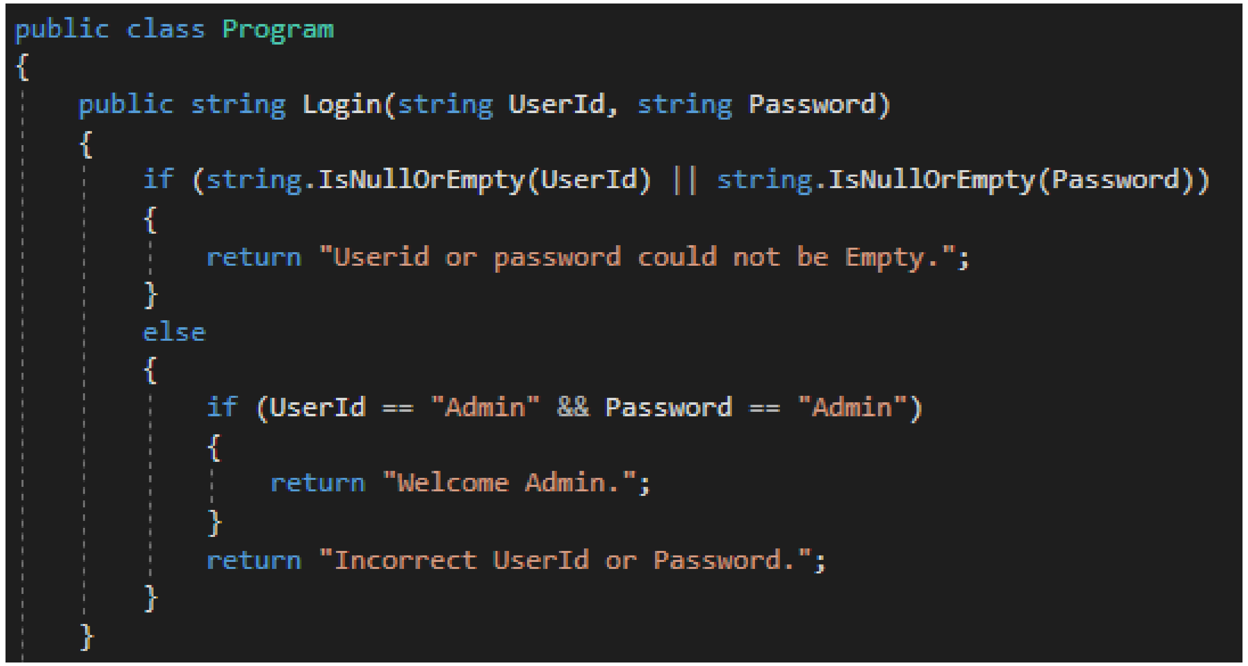
Create a class (EmployeeDetails.cs) and write this code:



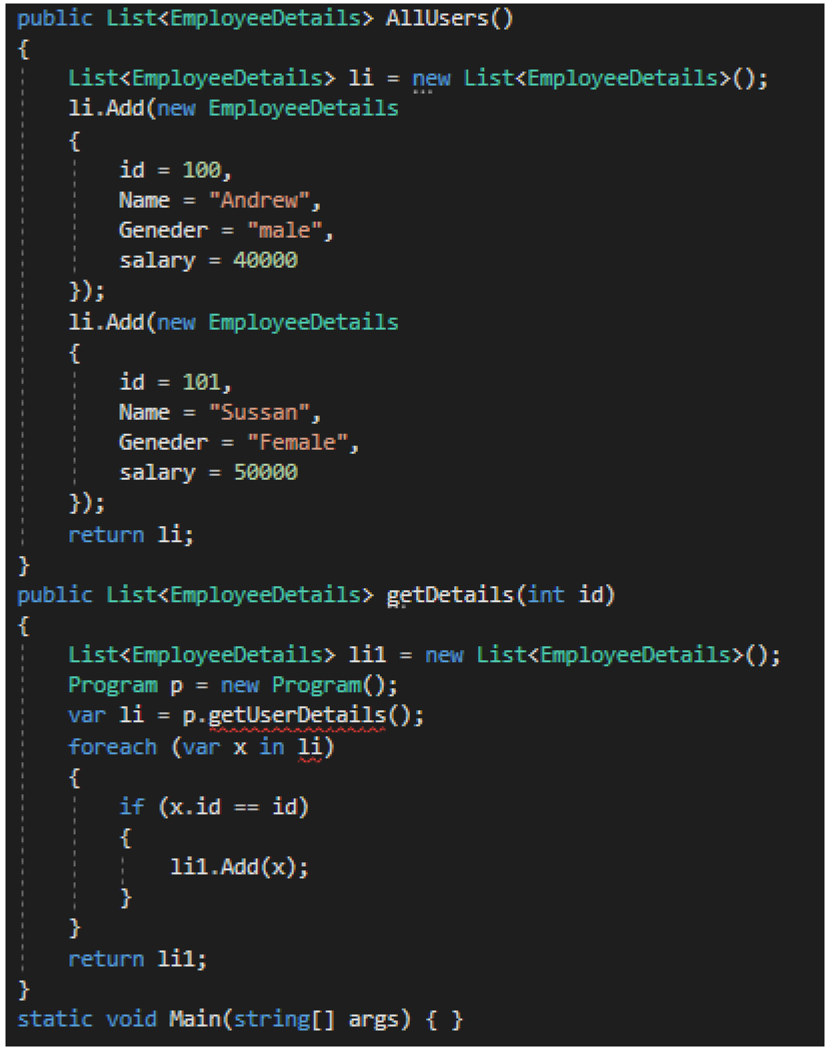
```
namespace DemoProject
{
    public class EmployeeDetails
    {
        public int id {get; set;}
        public string Name {get; set;}
        public double salary{get; set;}
        public string Geneder{get; set;}
    }
}
```

Unit Testing

Now in the class program write some conditions which need to be tested.



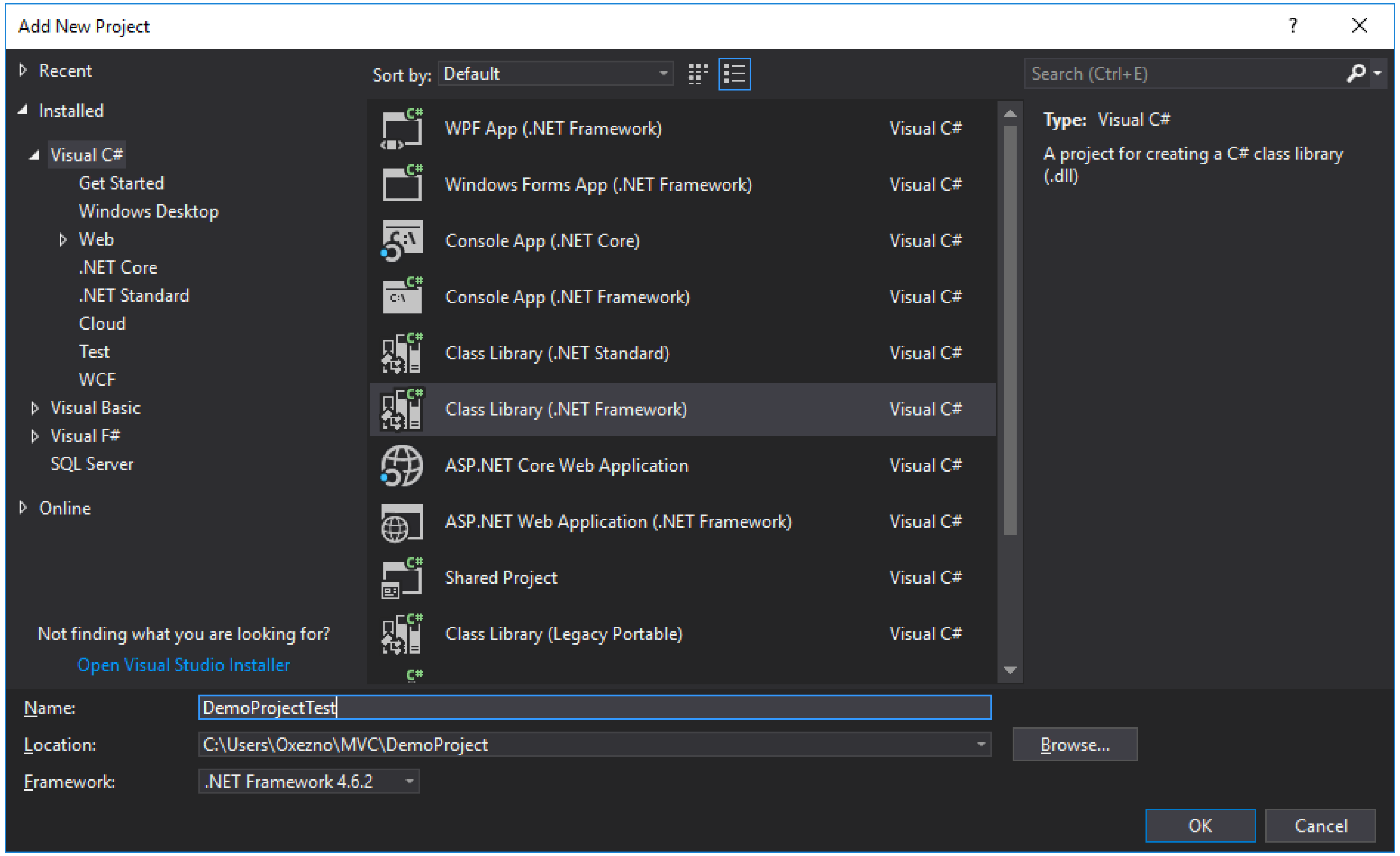
```
public class Program
{
    public string Login(string UserId, string Password)
    {
        if (string.IsNullOrEmpty(UserId) || string.IsNullOrEmpty(Password))
        {
            return "Userid or password could not be Empty.";
        }
        else
        {
            if (UserId == "Admin" && Password == "Admin")
            {
                return "Welcome Admin.";
            }
            return "Incorrect UserId or Password.";
        }
    }
```



```
public List<EmployeeDetails> AllUsers()
{
    List<EmployeeDetails> li = new List<EmployeeDetails>();
    li.Add(new EmployeeDetails
    {
        id = 100,
        Name = "Andrew",
        Geneder = "male",
        salary = 40000
    });
    li.Add(new EmployeeDetails
    {
        id = 101,
        Name = "Sussan",
        Geneder = "Female",
        salary = 50000
    });
    return li;
}
public List<EmployeeDetails> getDetails(int id)
{
    List<EmployeeDetails> li1 = new List<EmployeeDetails>();
    Program p = new Program();
    var li = p.getUserDetails();
    foreach (var x in li)
    {
        if (x.id == id)
        {
            li1.Add(x);
        }
    }
    return li1;
}
static void Main(string[] args) { }
```

Unit Testing

So please add a new project (Class Library Type) in the Solution Explorer as follows.



Add New Project

Recent
Installed
Visual C#
Get Started
Windows Desktop
Web
.NET Core
.NET Standard
Cloud
Test
WCF
Visual Basic
Visual F#
SQL Server
Online

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

Search (Ctrl+E)

WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Standard)	Visual C#
Class Library (.NET Framework)	Visual C#
ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#
Shared Project	Visual C#
Class Library (Legacy Portable)	Visual C#

Type: Visual C#

A project for creating a C# class library (.dll)

Name: DemoProjectTest

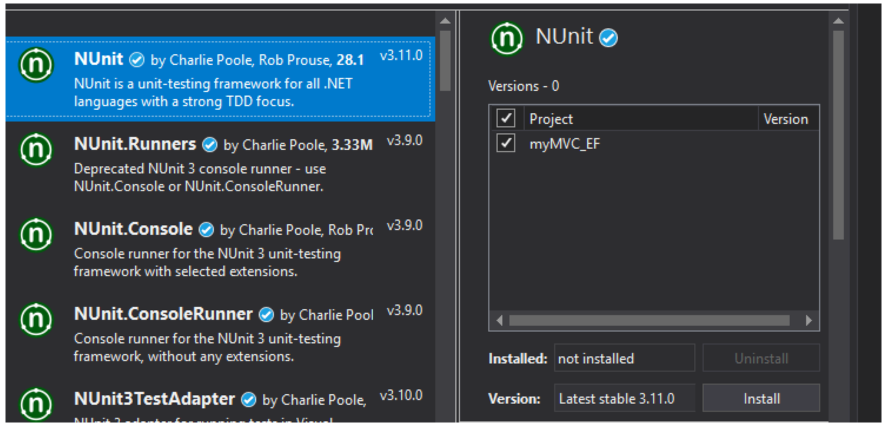
Location: C:\Users\Oxezno\MVC\DemoProject

Framework: .NET Framework 4.6.2

Browse... OK Cancel

Unit Testing

Install Nunit Nuget Package Manager.



NUnit ✓ by Charlie Poole, Rob Prouse, 28.1 v3.11.0
NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

NUnit.Runners ✓ by Charlie Poole, 3.33M v3.9.0
Deprecated NUnit 3 console runner - use NUnit.Console or NUnit.ConsoleRunner.

NUnit.Console ✓ by Charlie Poole, Rob Pro v3.9.0
Console runner for the NUnit 3 unit-testing framework with selected extensions.

NUnit.ConsoleRunner ✓ by Charlie Pool v3.9.0
Console runner for the NUnit 3 unit-testing framework, without any extensions.

NUnit3TestAdapter ✓ by Charlie Poole, v3.10.0

NUnit ✓

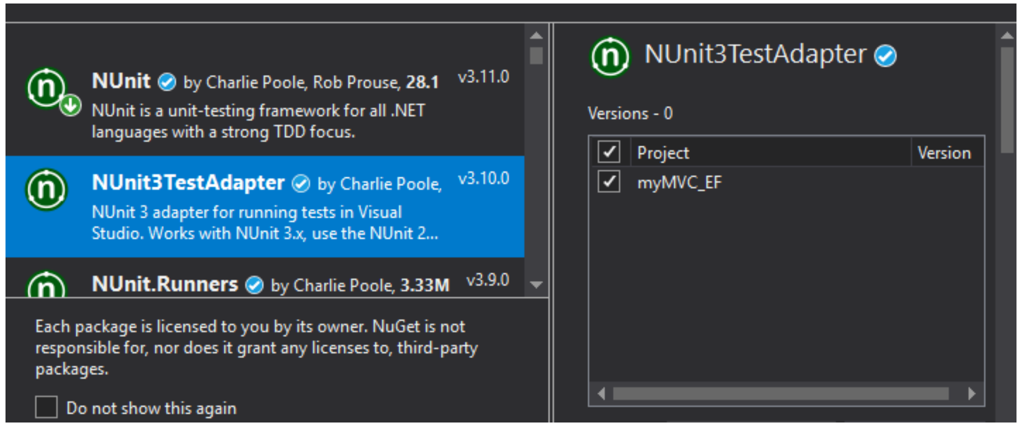
Versions - 0

	Project	Version
✓	myMVC_EF	

Installed: not installed — Uninstall

Version: Latest stable 3.11.0 — Install

Now after installing the Nunit we need one more dll that needs to be installed in the project.
NUnit Test Adapter for Visual Studio



NUnit ✓ by Charlie Poole, Rob Prouse, 28.1 v3.11.0
NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

NUnit3TestAdapter ✓ by Charlie Poole, v3.10.0
NUnit 3 adapter for running tests in Visual Studio. Works with NUnit 3.x, use the NUnit 2...

NUnit.Runners ✓ by Charlie Poole, 3.33M v3.9.0

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

☐ Do not show this again

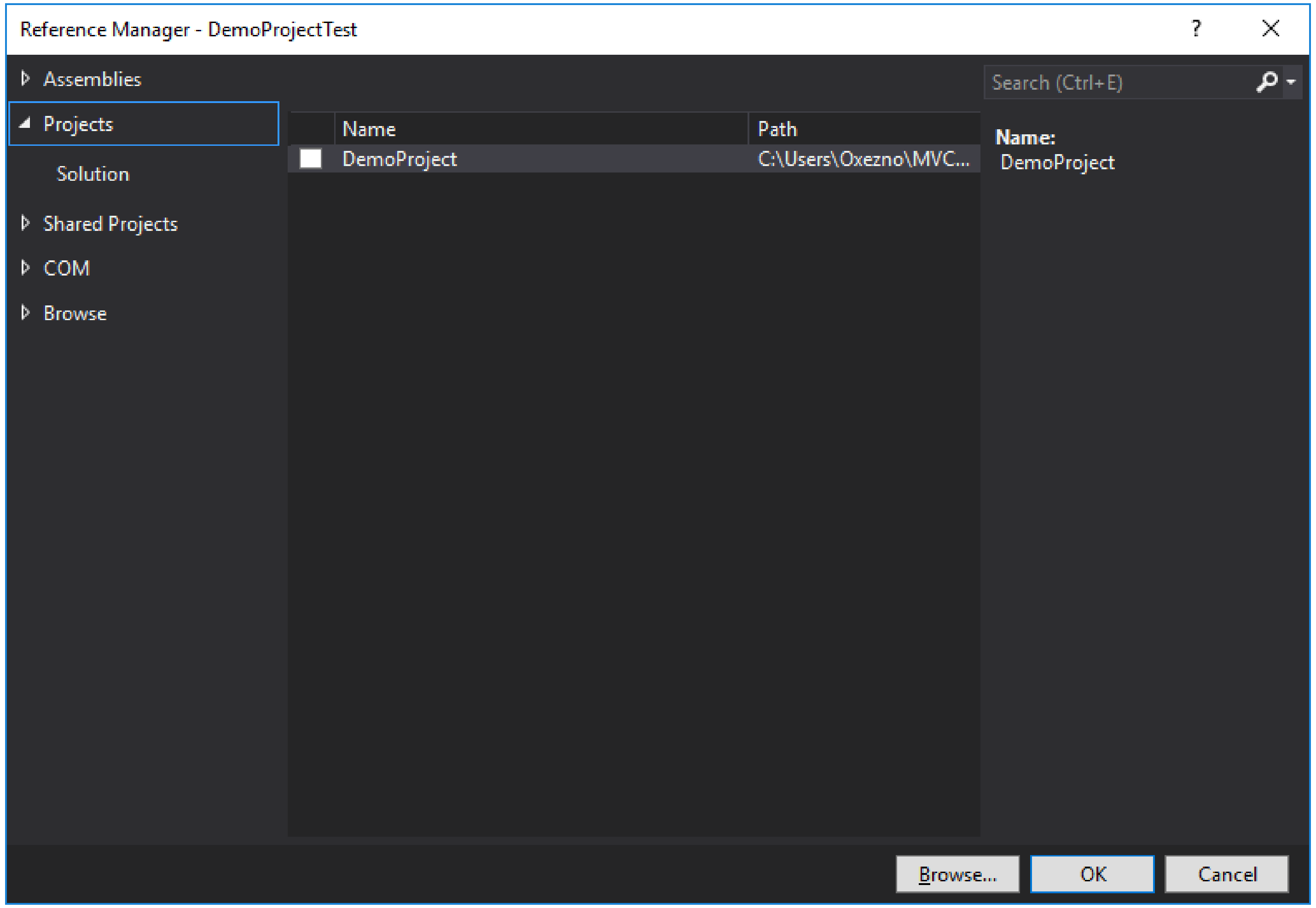
NUnit3TestAdapter ✓

Versions - 0

	Project	Version
✓	myMVC_EF	

Unit Testing

Add the Demo Project dll in DemoProjectTest References.

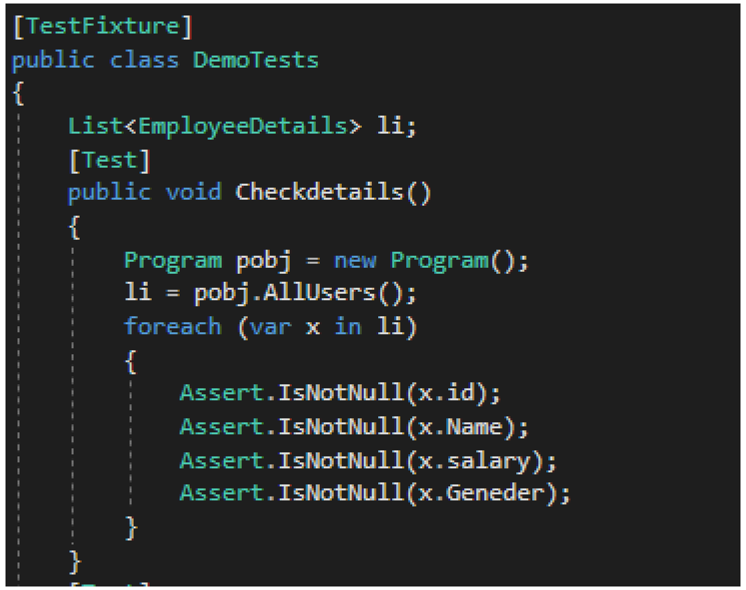


	Name	Path
☐	DemoProject	C:\Users\Oxezno\MVC...

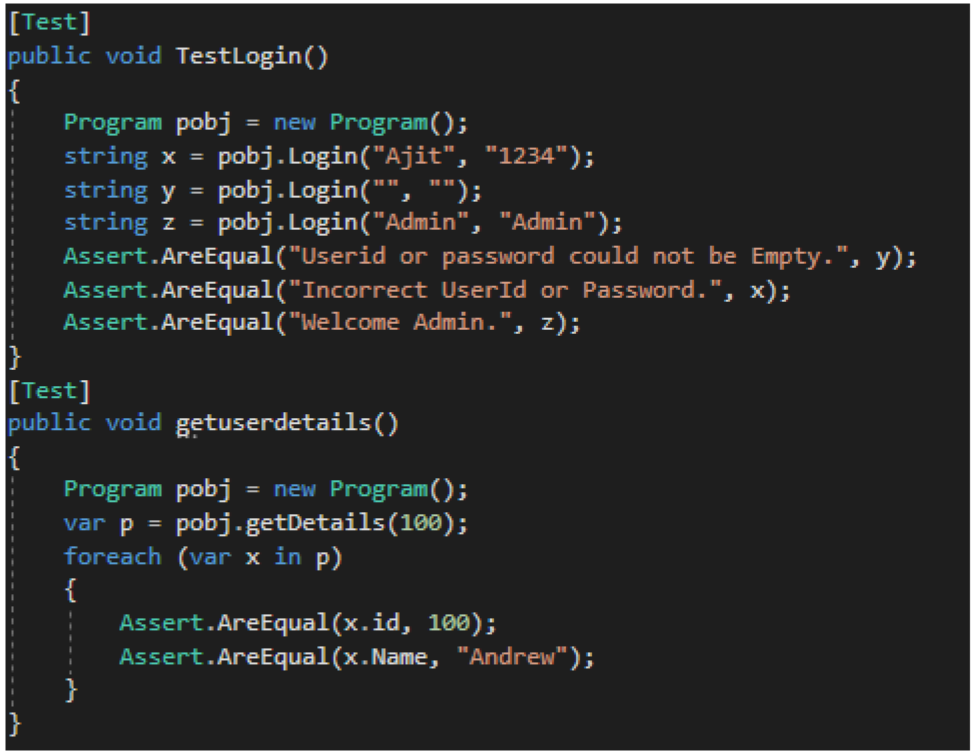
Name:
DemoProject

Unit Testing

Create a DemoTests.cs with the following code for testing:



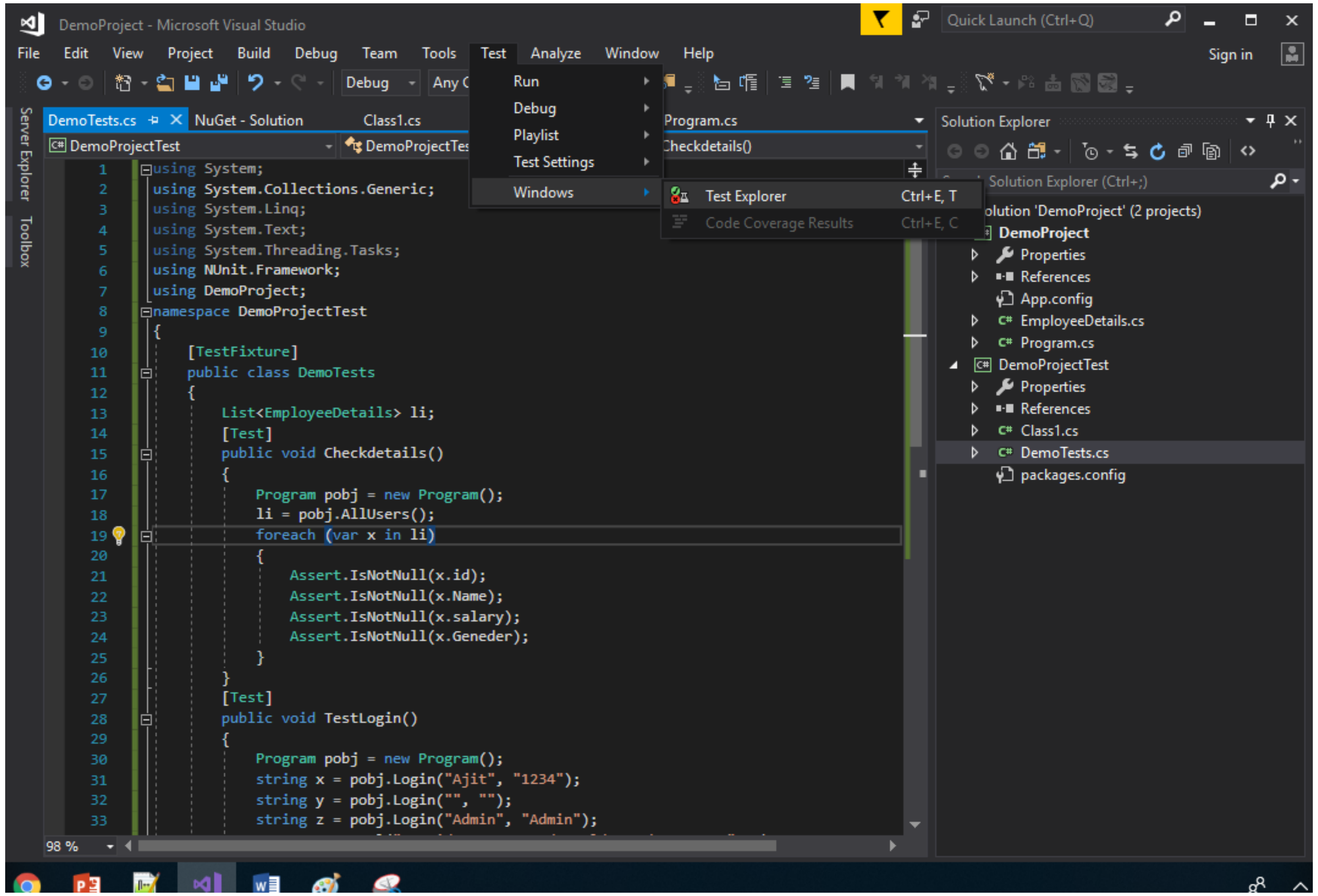
```
[TestFixture]
public class DemoTests
{
    List<EmployeeDetails> li;
    [Test]
    public void Checkdetails()
    {
        Program pobj = new Program();
        li = pobj.AllUsers();
        foreach (var x in li)
        {
            Assert.IsNotNull(x.id);
            Assert.IsNotNull(x.Name);
            Assert.IsNotNull(x.salary);
            Assert.IsNotNull(x.Geneder);
        }
    }
```



```
[Test]
public void TestLogin()
{
    Program pobj = new Program();
    string x = pobj.Login("Ajit", "1234");
    string y = pobj.Login("", "");
    string z = pobj.Login("Admin", "Admin");
    Assert.AreEqual("Userid or password could not be Empty.", y);
    Assert.AreEqual("Incorrect UserId or Password.", x);
    Assert.AreEqual("Welcome Admin.", z);
}
[Test]
public void getuserdetails()
{
    Program pobj = new Program();
    var p = pobj.getDetails(100);
    foreach (var x in p)
    {
        Assert.AreEqual(x.id, 100);
        Assert.AreEqual(x.Name, "Andrew");
    }
}
```

Unit Testing

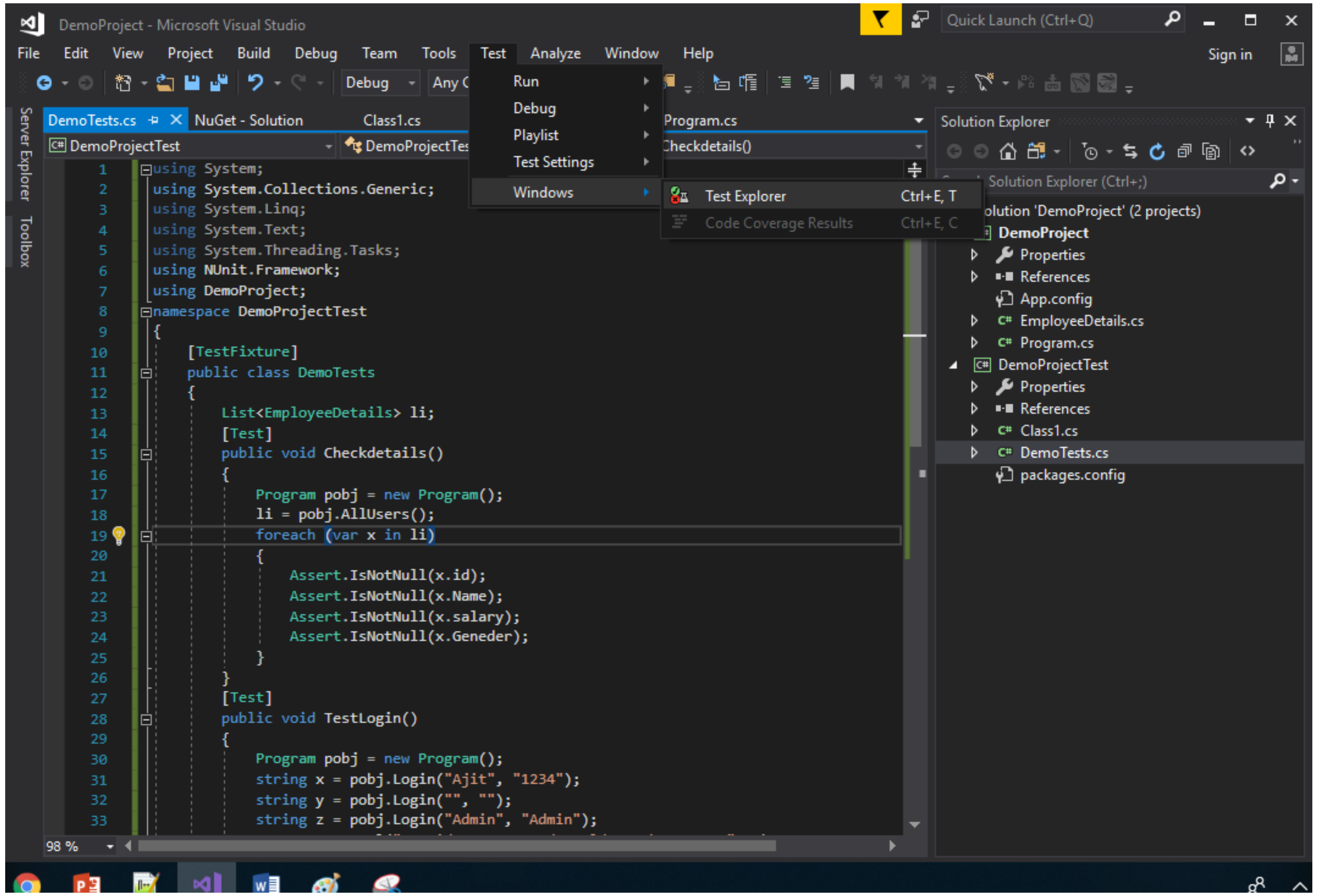
Now let's check our test. Go to the Visual Studio Test Explorer.



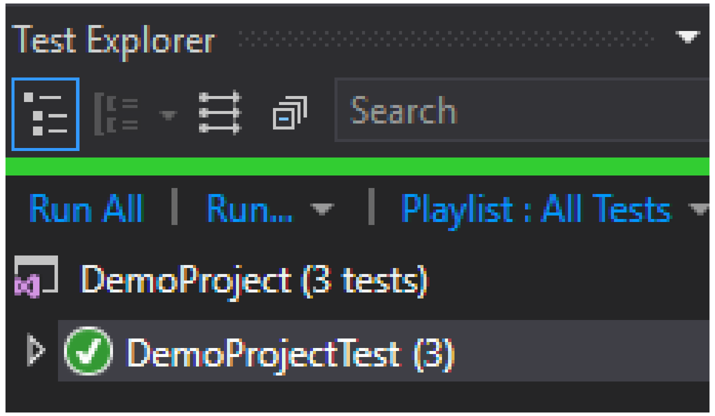
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using DemoProject;
namespace DemoProjectTest
{
    [TestFixture]
    public class DemoTests
    {
        List<EmployeeDetails> li;
        [Test]
        public void Checkdetails()
        {
            Program pobj = new Program();
            li = pobj.AllUsers();
            foreach (var x in li)
            {
                Assert.IsNotNull(x.id);
                Assert.IsNotNull(x.Name);
                Assert.IsNotNull(x.salary);
                Assert.IsNotNull(x.Geneder);
            }
        }
        [Test]
        public void TestLogin()
        {
            Program pobj = new Program();
            string x = pobj.Login("Ajit", "1234");
            string y = pobj.Login("", "");
            string z = pobj.Login("Admin", "Admin");
```

Unit Testing

Now let's check our test. Go to the Visual Studio Test Explorer.

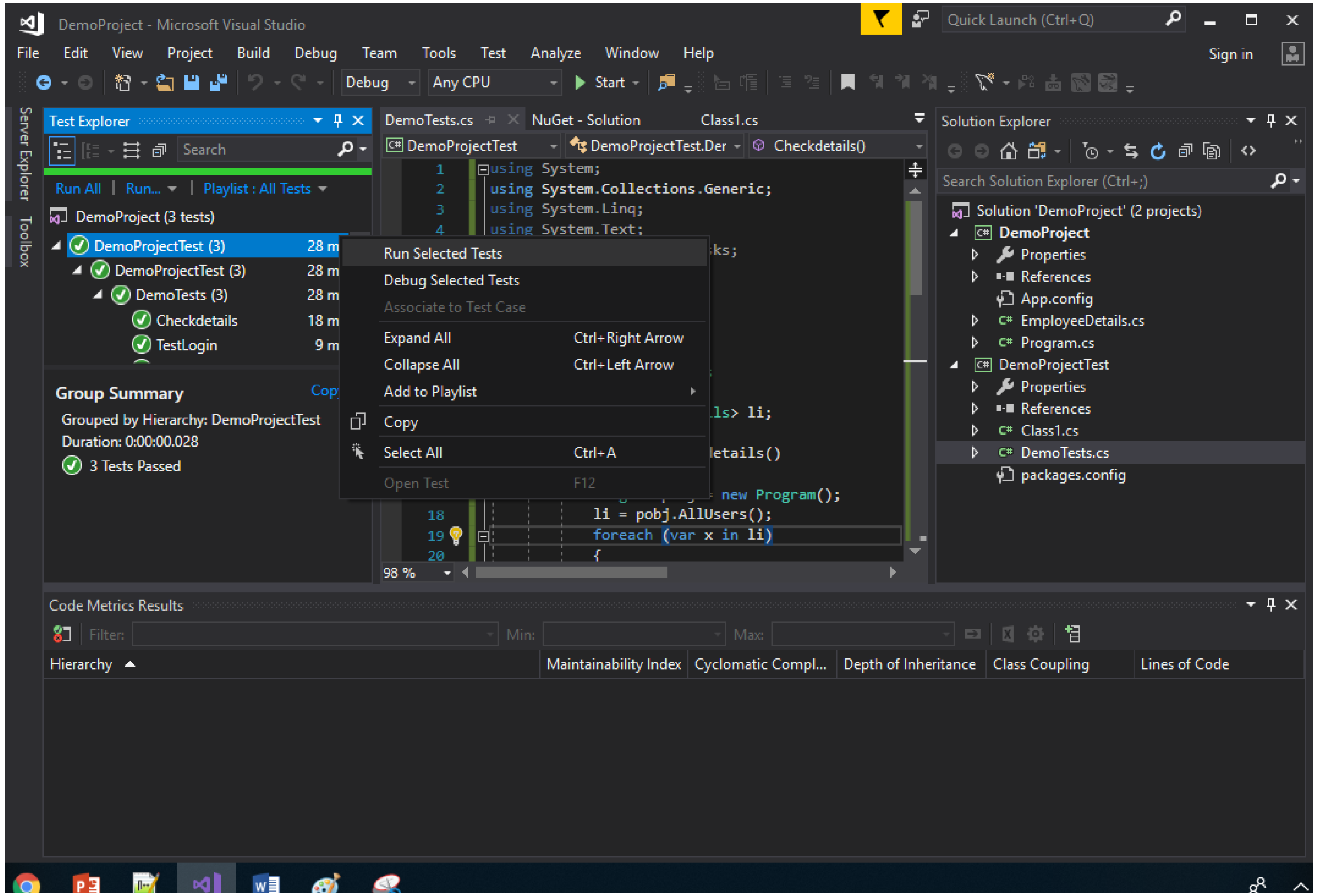


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using DemoProject;
namespace DemoProjectTest
{
    [TestFixture]
    public class DemoTests
    {
        List<EmployeeDetails> li;
        [Test]
        public void Checkdetails()
        {
            Program pobj = new Program();
            li = pobj.AllUsers();
            foreach (var x in li)
            {
                Assert.IsNotNull(x.id);
                Assert.IsNotNull(x.Name);
                Assert.IsNotNull(x.salary);
                Assert.IsNotNull(x.Geneder);
            }
        }
        [Test]
        public void TestLogin()
        {
            Program pobj = new Program();
            string x = pobj.Login("Ajit", "1234");
            string y = pobj.Login("", "");
            string z = pobj.Login("Admin", "Admin");
```



Unit Testing

Now from here if we run a test individually just right click on it and run the test

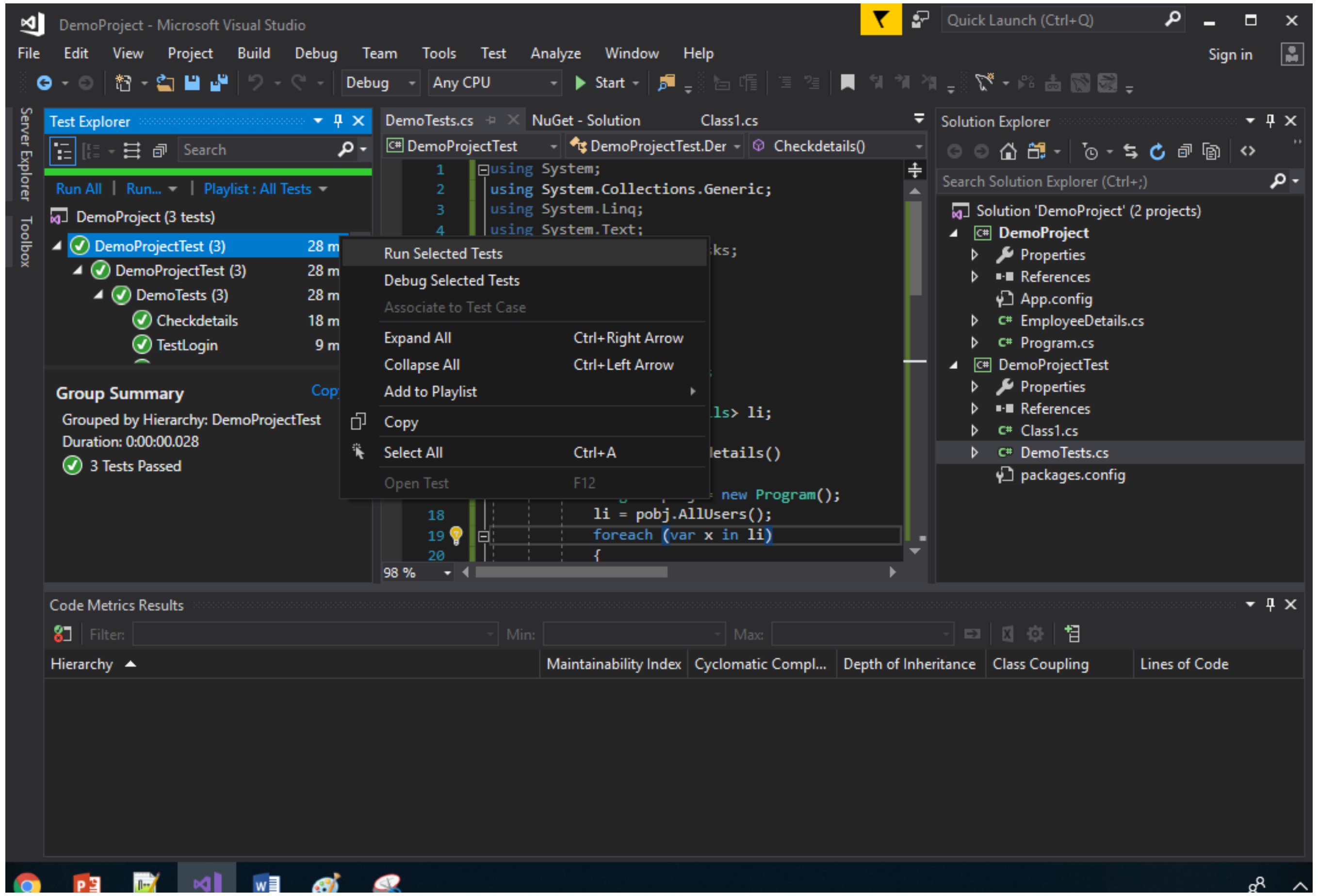


Group Summary
Grouped by Hierarchy: DemoProjectTest
Duration: 0:00:00.028
✓ 3 Tests Passed

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
ks;
ls> li;
etails()
      new Program();
      li = pobj.AllUsers();
      foreach (var x in li)
      {
```

Unit Testing

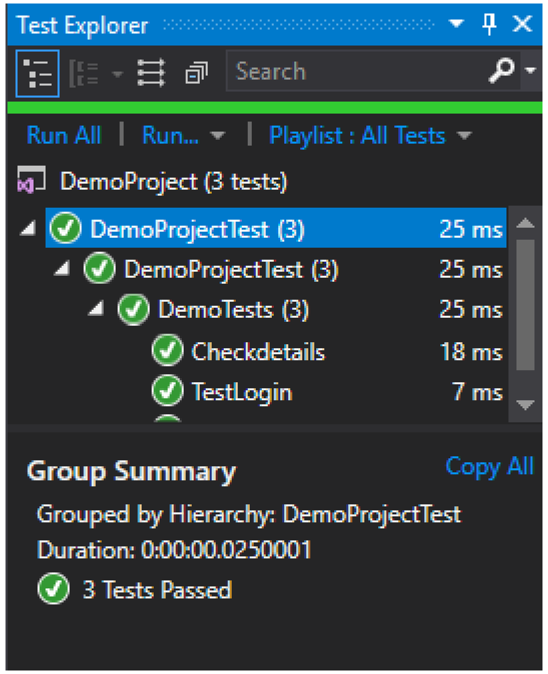
Now from here if we run a test individually just right click on it and run the test



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
DemoProject (3 tests)
  DemoProjectTest (3)    28 m
    DemoProjectTest (3)  28 m
      DemoTests (3)      28 m
        Checkdetails     18 m
        TestLogin         9 m

Group Summary
Grouped by Hierarchy: DemoProjectTest
Duration: 0:00:00.028
3 Tests Passed
```

```
Solution 'DemoProject' (2 projects)
  DemoProject
    Properties
    References
    App.config
    EmployeeDetails.cs
    Program.cs
  DemoProjectTest
    Properties
    References
    Class1.cs
    DemoTests.cs
    packages.config
```



```
DemoProject (3 tests)
  DemoProjectTest (3)    25 ms
    DemoProjectTest (3)  25 ms
      DemoTests (3)      25 ms
        Checkdetails     18 ms
        TestLogin         7 ms

Group Summary
Grouped by Hierarchy: DemoProjectTest
Duration: 0:00:00.0250001
3 Tests Passed
```