Contents

[Passing Data 2](#_Toc195563736)

[Query Parameters 2](#_Toc195563737)

[Route Parameters (URL Segments) 3](#_Toc195563738)

[Form Parameters (POST data) 3](#_Toc195563739)

[TempData / ViewData / ViewBag 4](#_Toc195563740)

[ViewBag / ViewData (same request only): 4](#_Toc195563741)

[Session Variables 5](#_Toc195563742)

[Cookies 5](#_Toc195563743)

[Web API More Topics 6](#_Toc195563744)

[Creating a Web API Application 6](#_Toc195563745)

[Adding Bearer Authentication to Web API 6](#_Toc195563746)

[Connecting SQL with Web API 10](#_Toc195563747)

[Implementing Entity Framework 11](#_Toc195563748)

[Testing Web API using Postman 12](#_Toc195563749)

[Deploying an ASP.NET Web Application (.NET Framework) to Azure App Service 12](#_Toc195563750)

# Passing Data

## Query Parameters

**Example URL**

http://example.com/Home/Search?query=aspnet&page=2

**Controller action:**

public class HomeController : Controller

{

public ActionResult Search(string query, int page = 1)

{

// query = "aspnet"

// page = 2

return View();

}

}

* query and page are automatically bound from the query string.
* Default values like int page = 1 are supported.

**Accessing Query Parameters Manually**

You can also access query parameters directly from Request.QueryString:

public ActionResult Search()

{

string query = Request.QueryString["query"];

string page = Request.QueryString["page"];

// Convert if needed

int pageNumber = string.IsNullOrEmpty(page) ? 1 : Convert.ToInt32(page);

return View();

}

**Optional Parameters**

You can make parameters optional by assigning default values or using nullable types:

public ActionResult Details(int? id)

{

if (id == null)

{

return HttpNotFound();

}

// Proceed with id.Value

return View();

}

## Route Parameters (URL Segments)

Instead of query strings like ?id=5, you can embed values directly in the URL using route parameters.

// RouteConfig.cs

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

// Controller:

public ActionResult Details(int id)

{

// id comes from the URL: /Home/Details/5

return View();

}

## Form Parameters (POST data)

When submitting a form using POST, data is sent in the form body, not the URL.

View (Razor):

@using (Html.BeginForm("SubmitForm", "Home", FormMethod.Post))

{

<input type="text" name="username" />

<input type="submit" value="Submit" />

}

Controller:

[HttpPost]

public ActionResult SubmitForm(string username)

{

// username comes from form data

return View();

}

## TempData / ViewData / ViewBag

* These are used for short-term data storage between actions or between controller and view.
* Used to pass data between actions (survives a redirect, single-use only).

**Example 1**

public ActionResult RedirectWithData()

{

TempData["UserId"] = 123;

return RedirectToAction("Profile");

}

public ActionResult Profile()

{

int userId = (int)TempData["UserId"];

return View();

}

**Example 2**

public ActionResult ActionA()

{

TempData["Message"] = "Hello from ActionA!";

return RedirectToAction("ActionB");

}

public ActionResult ActionB()

{

string message = TempData["Message"] as string;

ViewBag.Message = message;

return View();

}

## ViewBag / ViewData (same request only):

**Example 1 (ViewData)**

Used to pass data from controller to view, only for the current request.

public ActionResult MyView()

{

ViewData["Title"] = "Welcome Page";

return View();

}

View:

<h1>@ViewData["Title"]</h1>

**Example 2 (ViewBag)**

public ActionResult About()

{

ViewBag.Company = "Contoso Ltd.";

return View();

}

View:

<p>Company: @ViewBag.Company</p>

## Session Variables

Used to store data across multiple requests and actions (per user/session).

// Set Session:

public ActionResult Login()

{

Session["UserId"] = 101;

return RedirectToAction("Dashboard");

}

// Get Session:

public ActionResult Dashboard()

{

int? userId = Session["UserId"] as int?;

ViewBag.UserId = userId;

return View();

}

## Cookies

Used to store client-side data that persists across requests and sessions.

// Set Cookie:

public ActionResult SetCookie()

{

HttpCookie cookie = new HttpCookie("UserTheme");

cookie.Value = "DarkMode";

cookie.Expires = DateTime.Now.AddDays(1);

Response.Cookies.Add(cookie);

return RedirectToAction("ReadCookie");

}

// Read Cookie:

public ActionResult ReadCookie()

{

string theme = Request.Cookies["UserTheme"]?.Value;

ViewBag.Theme = theme;

return View();

}

# Web API More Topics

## Creating a Web API Application

Steps:

1. Open Visual Studio → New Project
2. Choose ASP.NET Web Application (.NET Framework)
3. Select Web API template

Example Controller:

public class HelloController : ApiController

{

[HttpGet]

public string GetGreeting()

{

return "Hello, Web API!";

}

}

URL to test: http://localhost:[port]/api/hello

## Adding Bearer Authentication to Web API

This uses OAuth 2.0 with bearer tokens via OWIN middleware.

Required Packages (via NuGet)

Install these via Package Manager Console:

bash

Copy

Edit

Install-Package Microsoft.Owin.Host.SystemWeb

Install-Package Microsoft.Owin.Security.OAuth

Install-Package Microsoft.Owin.Cors

Install-Package Owin

📁 File Structure

diff

Copy

Edit

- App\_Start/

- Controllers/

- SecureController.cs

- Providers/

- SimpleAuthorizationServerProvider.cs

- Startup.cs

- Web.config

🛠 Step-by-Step Implementation

🔹 1. Startup.cs

csharp

Copy

Edit

using Microsoft.Owin;

using Microsoft.Owin.Security.OAuth;

using Owin;

using System;

using WebAPIAuth.Providers;

[assembly: OwinStartup(typeof(WebAPIAuth.Startup))]

namespace WebAPIAuth

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

var OAuthOptions = new OAuthAuthorizationServerOptions

{

AllowInsecureHttp = true, // Disable in production

TokenEndpointPath = new PathString("/token"),

AccessTokenExpireTimeSpan = TimeSpan.FromHours(1),

Provider = new SimpleAuthorizationServerProvider()

};

app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll); // Allow CORS

app.UseOAuthAuthorizationServer(OAuthOptions);

app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());

}

}

}

🔹 2. Providers/SimpleAuthorizationServerProvider.cs

csharp

Copy

Edit

using Microsoft.Owin.Security.OAuth;

using System.Security.Claims;

using System.Threading.Tasks;

namespace WebAPIAuth.Providers

{

public class SimpleAuthorizationServerProvider : OAuthAuthorizationServerProvider

{

public override async Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)

{

context.Validated(); // No client validation for demo

}

public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)

{

// Hardcoded for demo - replace with DB lookup in production

if (context.UserName == "admin" && context.Password == "password")

{

var identity = new ClaimsIdentity(context.Options.AuthenticationType);

identity.AddClaim(new Claim(ClaimTypes.Name, context.UserName));

identity.AddClaim(new Claim(ClaimTypes.Role, "Admin"));

context.Validated(identity);

}

else

{

context.SetError("invalid\_grant", "Username or password is incorrect.");

}

}

}

}

🔹 3. Controllers/SecureController.cs

csharp

Copy

Edit

using System.Web.Http;

namespace WebAPIAuth.Controllers

{

[Authorize]

[RoutePrefix("api/secure")]

public class SecureController : ApiController

{

[HttpGet]

[Route("hello")]

public IHttpActionResult GetHello()

{

var user = User.Identity.Name;

return Ok("Hello, " + user + "! This is a protected resource.");

}

}

}

🔹 4. Enable Web API Routes (WebApiConfig.cs)

csharp

Copy

Edit

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

🚀 Testing with Postman

🔸 Step 1: Request a Token

Method: POST

URL: http://localhost:port/token

Headers:

Content-Type: application/x-www-form-urlencoded

Body (x-www-form-urlencoded):

ini

Copy

Edit

username=admin

password=password

grant\_type=password

Response:

json

Copy

Edit

{

"access\_token": "eyJhbGciOi...",

"token\_type": "bearer",

"expires\_in": 3600

}

🔸 Step 2: Access Secured API

Method: GET

URL: http://localhost:port/api/secure/hello

Headers:

makefile

Copy

Edit

Authorization: Bearer eyJhbGciOi...

Response:

json

Copy

Edit

"Hello, admin! This is a protected resource."

## Connecting SQL with Web API

Sample Web.config connection string:

<connectionStrings>

<add name="DefaultConnection"

connectionString="Data Source=.;Initial Catalog=WebApiDb;Integrated Security=True"

providerName="System.Data.SqlClient"/>

</connectionStrings>

## Implementing Entity Framework

Use Code First approach.

**Install Entity Framework:**

Install-Package EntityFramework

**Create Model and Context:**

Models/Product.cs

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

Data/ApplicationDbContext.cs

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext() : base("DefaultConnection") { }

public DbSet<Product> Products { get; set; }

}

Create API Controller:

public class ProductsController : ApiController

{

private ApplicationDbContext db = new ApplicationDbContext();

[HttpGet]

public IEnumerable<Product> Get()

{

return db.Products.ToList();

}

[HttpPost]

public IHttpActionResult Post(Product product)

{

db.Products.Add(product);

db.SaveChanges();

return Ok(product);

}

}

## Testing Web API using Postman

GET Request:

GET http://localhost:[port]/api/products

POST Request:

POST http://localhost:[port]/api/products

Body (JSON):

{

"Name": "Laptop",

"Price": 999.99

}

Headers:

Content-Type: application/json

Authorization: Bearer [your\_token\_here]

# Deploying an ASP.NET Web Application (.NET Framework) to Azure App Service

**Deploy via Visual Studio (Recommended)**

Prerequisites:

* Visual Studio with Azure SDK installed
* Azure Subscription
* ASP.NET Web App (e.g., .NET Framework Web API or MVC app)

Step-by-Step (Visual Studio):

1. Right-click your project → Publish

1. Choose: Azure → Azure App Service (Windows)
2. Click Create a new App Service

2. Create App Service (if not existing)

1. Name: mywebapp123
2. Resource Group: Create or choose existing
3. Hosting Plan: Choose region and pricing tier
4. Click Create

3. Deploy

* Visual Studio will build the app, publish it, and open it in your browser
* Your app is now live at:

https://mywebapp123.azurewebsites.net