# Walkthrough 12 - Introduction to Web API

## Setup

This lab will explore Web API.

1. Start SQL Server.
2. Start Visual Studio.
3. Click Create a new project.
4. Set language to **C#** and project type to **Web**.
5. Select the ASP.NET Core Web API template, click Next.
6. Set Project name to **MedicationAPI**.
7. Set Location to **a folder of your choosing**.
8. Ensure Place solution and project in the same directory is unchecked, click Next.
9. Set version to **.NET 5.0**, set Authentication Type to **None**, unselect Configure for HTTPS, select Enable OpenAPI support, if necessary, click Create.

## WeatherForecastController.cs

1. Open Controllers / WeatherForecastController.cs
2. Notice the Get method in the controller.
3. Press Ctrl+F5 to run the site, the Swagger documentation page will appear, more on Swagger later.
4. Change the URL to **http://localhost:12345/weatherforecast**; notice that JSON is returned instead of a web page.
5. Delete WeatherForecastController.cs and WeatherForecast.cs.

## appsettings.json

1. Add a connection string for the CHDB database.
2. 
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CHDB": "Server=localhost\\sqlexpress;Database=CHDB;Trusted_Connection=True"
  }
}
```
3. Save the file.

## Scaffold-DbContext

1. Open the Package Manager Console (PMC) and issue the following commands.
2. **Install-Package Microsoft.EntityFrameworkCore.Tools**
3. **Install-Package Microsoft.EntityFrameworkCore.SqlServer**
4. **Scaffold-DbContext name=chdb Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Tables medications**

## Startup.cs

1. Register the database context. Add the **using MedicationAPI.Models;** and **using Microsoft.EntityFrameworkCore;** directives.
2. 
```
public void ConfigureServices(IServiceCollection services)
{

    services.AddControllers();
    services.AddDbContext<CHDBContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("CHDB")));
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "MedicationAPI", Version = "v1" });
    });
}
```
3. Notice that no default routing has been specified in the Configure method; the endpoints will be specified in the controller.
4. 
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```
5. Save the file.

## MedicationsController.cs

1. Right-click Controllers folder and select Add / Controller... .
2. Select the API templates, select API Controller with actions, using Entity Framework, click Add.
3. Set Model class to **Medication (Medication.Models)**.
4. Set Data context class to **CHDBContext (Medication.Models)**.
5. Accept default name of MedicationsController, click Add.
6. Run the site, navigate to **http://localhost:12345/api/medications** to see all medications.
7. Navigate to **http://localhost:12345/api/medications/10** to see an individual medication.
8. Update the routing to not use the word api.
9. 
```
[Route("~~api/~~[controller]")]
[ApiController]
public class MedicationsController : ControllerBase
{
    private readonly CHDBContext _context;
    ...
```
10. Save the file.
11. Refresh the browser, the data won't be found. Navigate to **http://localhost:12345/medications/11** to see a medication.
12. Change the routing to **meds**.
13. 
```
[Route("~~[controller]~~meds")]
[ApiController]
public class MedicationsController : ControllerBase
{
    private readonly CHDBContext _context;
    ...
```
14. Save the file.
15. Refresh the browser, the data won't be found. Navigate to **http://localhost:12345/meds/12** to see a medication.
16. Change the routing back.
17. 
```
[Route("~~meds~~[controller]")]
[ApiController]
public class MedicationsController : ControllerBase
{
    private readonly CHDBContext _context;
    ...
```
18. Save the file.
19. Refresh the browser, the data won't be found. Navigate to **http://localhost:12345/medications/13** to see a medication.
20. Change the class name to **Temp**, also change the constructor name.
21. 
```
[Route("[controller]")]
[ApiController]
public class ~~MedicationsController~~Temp : ControllerBase
{
    private readonly CHDBContext _context;

    public ~~MedicationsController~~Temp(CHDBContext context)
    {
        _context = context;
    }
    ...
```
22. Save the file, navigate to **http://localhost:12345/temp/14** to see a medication.
23. Change the name back to **MedicationsController**.
24. 
```
[Route("[controller]")]
[ApiController]
public class ~~Temp~~MedicationsController : ControllerBase
{
    private readonly CHDBContext _context;

    public ~~Temp~~MedicationsController(CHDBContext context)
    {
        _context = context;
    }
    ...
```
25. Save the file, navigate to **http://localhost:12345/medications/15** to see a medication.
26. Change the GetMedication method name to **Temp**.
27. 
```
[HttpGet("{id}")]
public async Task<ActionResult<Medication>> ~~GetMedication~~Temp(int id)
{
...
```
28. Save the file.
29. Refresh the browser, the data remains, because the routing hasn't changed.
30. Change the method name back to **GetMedication**.
31. 
```
[HttpGet("{id}")]
public async Task<ActionResult<Medication>> ~~Temp~~GetMedication(int id)
{
...
```

32. Save the file.
33. Refresh the browser, the data remains.
34. Close the browser.

## launchSettings.json

1. In Solution Explorer, expand Properties and select launchSettings.json.
2. Update the launchUrl to **medications**.
3. 
```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:4105",
      "sslPort": 0
    }
  },
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "~~swagger~~medications",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "MedicationAPI": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "~~swagger~~medications",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "dotnetRunMessages": "true",
      "applicationUrl": "http://localhost:5000"
    }
  }
}
```
4. Save the file.
5. Run the site, the medications endpoint is served.

## Swagger

1. Navigate to **http://localhost:12345/swagger**.
2. Under the MedicationAPI heading, click the /swagger/v1/swagger.json link to see the generated documentation.
3. In the Schemas section, expand Medication to see the JSON representation of the Medication class.

## MedicationsController

1. Highlight all of the DeleteMedication method. Press Ctrl+K,Ctrl+C to comment it.
2. 
```
//[HttpDelete("{id}")]
//public async Task<IActionResult> DeleteMedication(int id)
//{
//    var medication = await _context.Medications.FindAsync(id);
//    if (medication == null)
//    {
//        return NotFound();
//    }

//    _context.Medications.Remove(medication);
//    await _context.SaveChangesAsync();

//    return NoContent();
//}
```
3. Save the file.
4. Refresh the browser, notice that the Delete verb is no longer documented.
5. If necessary, highlight all of the DeleteMedication method. Press Ctrl+K,Ctrl+U to uncomment it.

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteMedication(int id)
{
    var medication = await _context.Medications.FindAsync(id);
    if (medication == null)
    {
        return NotFound();
    }

    _context.Medications.Remove(medication);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

1. Save the file.

## Swagger

1. Refresh the browser, notice that the Delete verb is documented again.

2. Click the GET button with /Medications. Click the Try it out button, click the Execute button. The status code 200 will be returned along with all medications in JSON. Click the GET button again to collapse the section.
3. Click the GET button with /MedicationAPI/{id}. Click the Try it out button, click the Execute button. Notice that the id parameter is required. Set it to **1** and click Execute. Only the 1st medication is returned.
4. Change the id parameter to **200**, click Execute. This medication doesn't exist, so 404 is returned.
5. Change the id parameter back to **1**, click Execute. Select the Response body and copy it (Control+C). Collapse the section.
6. Click the PUT button. Click Try it out. Set id to **1**.
7. Select the template Request body and delete it. Paste in the copied body (Control+V) and change the description to **Advil**.
8. 
```
{
  "medicationId": 1,
  "medicationDescription": "~~Test Med~~Advil",
  "medicationCost": 1.23,
  "packageSize": "50 Tablets",
  "strength": "10 MG",
  "sig": "PRN",
  "unitsUsedYtd": 1231,
  "lastPrescribedDate": "2019-05-22T00:00:00"
}
```
9. Click Execute, 204 should be returned indicating success. Collapse the section.
10. Click GET /Medications/{id}. Click Execute, the updated medication will be returned. Collapse the section.
11. Click POST, click Try it out. Replace the Request body with the previously copied medication. Change the medication id to **170** and the description to **Ibuprofen**.
12. 
```
{
  "medicationId": 170,
  "medicationDescription": "~~Test Med~~Ibuprofen",
  "medicationCost": 1.23,
  "packageSize": "50 Tablets",
  "strength": "10 MG",
  "sig": "PRN",
  "unitsUsedYtd": 1231,
  "lastPrescribedDate": "2019-05-22T00:00:00"
}
```
13. Click Execute, 201 should be returned indicating success. Collapse the section.
14. Click GET /Medications. Click Clear, then Execute. Scroll to the bottom of the Response body to see the newly added medication. Collapse the section.
15. Click DELETE, click Try it out, set id to **170**, click Execute, 200 should be returned indicating success.
16. Click GET /Medications. Click Clear, then Execute. Scroll to the bottom of the Response body to see the newly added medication is now gone. Collapse the section.

## MedicationAPITest

1. Add a test project to the solution. Right-click the Solution and select Add / New Project... .
2. Set language to **C#** and project type to **Test**.
3. Select xUnit Test Project, click Next.
4. Set Name to **MedicationAPITest**, leave Location as is, click Next.
5. Set version to **.NET 5.0**, click Create.
6. Right-click the new project and select Add / Project Reference... .
7. Select MedicationAPI, click OK.
8. In the PMC, change the Default project drop-down list (near the top of the PMC) to **MedicationAPITest**.
9. Issue the command **Install-Package Microsoft.EntityFrameworkCore.InMemory**

## UnitTest1.cs

1. Initialize the in-memory database. Add the **using MedicationAPI.Models;**, **using Microsoft.EntityFrameworkCore;** directives.
2. 
```
public class UnitTest1
{
    private CHDBContext context;

    public UnitTest1()
    {
        var options = new DbContextOptionsBuilder<CHDBContext>().UseInMemoryDatabase(databaseName: "Medication_Tests").Options;
        context = new CHDBContext(options);

        context.Medications.AddRange(
            new Medication
            {
                MedicationId = 1,
                MedicationDescription = "Pain Reliever",
                MedicationCost = 1.0M
            },
            new Medication
            {
                MedicationId = 2,
                MedicationDescription = "Ibuprofen",
                MedicationCost = 1.25M
            },
            new Medication
            {
                MedicationId = 3,
                MedicationDescription = "Advil",
                MedicationCost = 1.5M
            }
        );

        context.SaveChanges();
    }

    [Fact]
    public void Test1()
    {

    }
}
```

3. Update the test method. Add the **using System.Threading.Tasks;**, **using MedicationAPI.Controllers;**, **using Microsoft.AspNetCore.Mvc;** and **using System.Collections.Generic;** directives.
4. 
```
[Fact]
public async ~~void~~Task ~~Test1~~Get_NoInput_ReturnsMedications()
{
    // Arrange
    var medicationsController = new MedicationsController(context);

    // Act
    var actionResult = await medicationsController.GetMedications();

    // Assert
    Assert.IsType<ActionResult<IEnumerable<Medication>>>(actionResult);
    var genericMedications = actionResult.Value;
    var medications = genericMedications as List<Medication>;
    Assert.Equal(3, medications.Count);
    Assert.Equal(1, medications[0].MedicationId);
    Assert.Equal("Ibuprofen", medications[1].MedicationDescription);
    Assert.Equal(1.5M, medications[2].MedicationCost);
}
```

5. Build the project, run the test.