Tutorial 9

Objectives

- Working with APIs
    - Asynchronous programming – Async and await
- JSON
- Make HTTP requests against our API
- Authentication
- Swagger

To follow this tutorial you first need to complete Week 7 and Week 8 tutorials

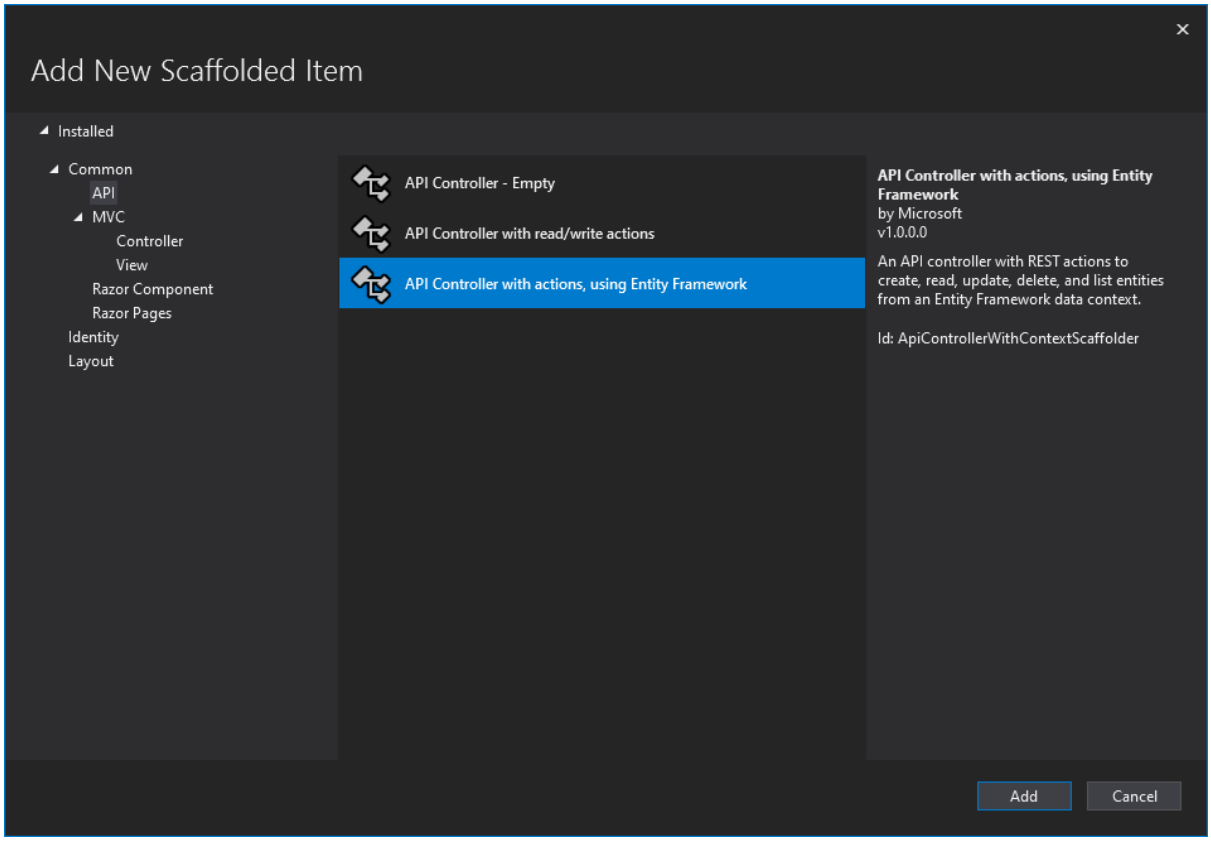
Prerequisites

- Todo model and CRUD pages from Week 7
- ApplicationDbContext from Week 8

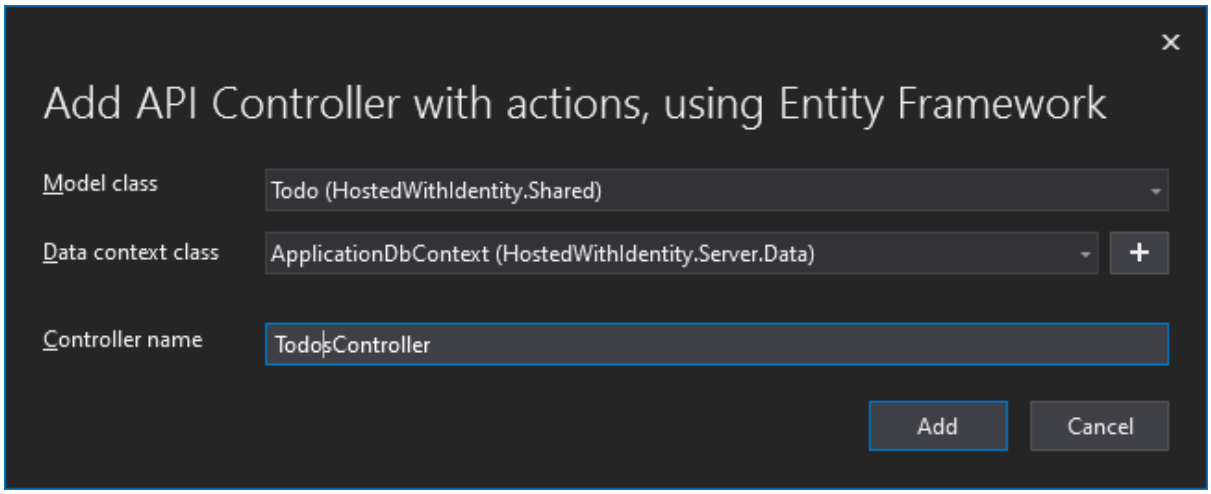
Working with APIs

Steps to create an API controller

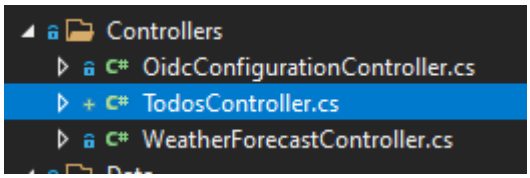
- Navigate to Server project > Controllers > right click on the Controllers folder > Add > Controller



Select Model class (Todo), Data context class (ApplicationDbContext) and give name to the controller.



Hint: If an error occurs, just click “Add” button again and it should be successfully completed.



Visual Studio will do everything for us, it will generate a Controller with CRUD operations.

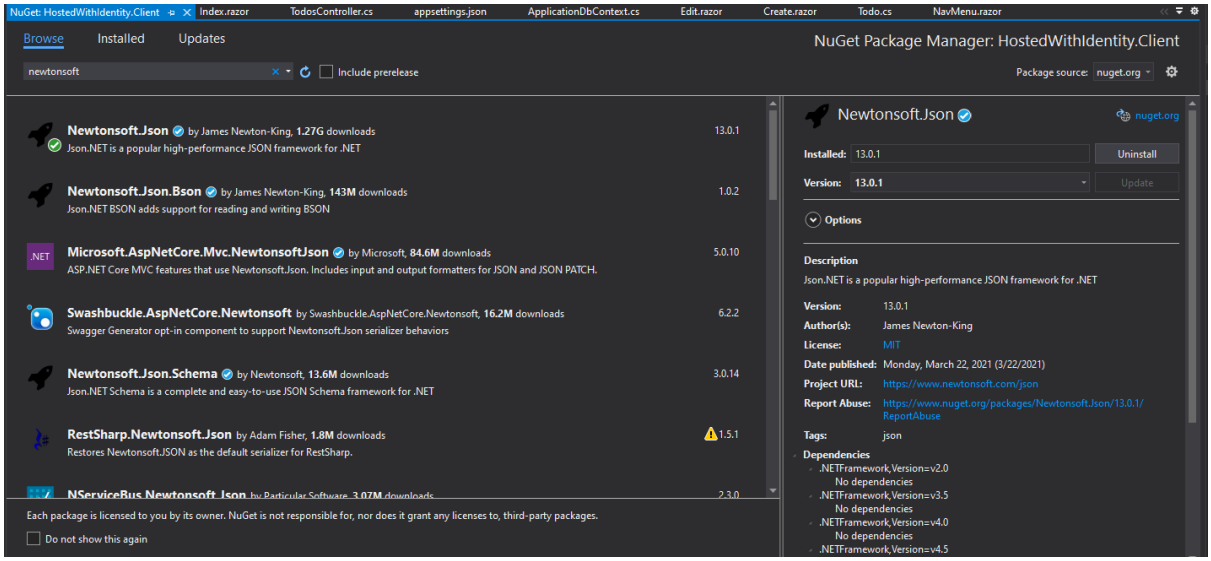
```
TodosController.cs   WeatherForecastController.cs   appsettings.json   ApplicationDbContext.cs   Index.razor   Edit.razor   Create.razor   Todo.cs
HostedWithIdentity.Server   HostedWithIdentity.Server.Controllers.TodosController   GetTodos()
7   using Microsoft.EntityFrameworkCore;
8   using HostedWithIdentity.Server.Data;
9   using HostedWithIdentity.Shared;
10
11  namespace HostedWithIdentity.Server.Controllers
12  {
13      [Route("api/[controller]")]
14      [ApiController]
        1 reference
15      public class TodosController : ControllerBase
16      {
17          private readonly ApplicationDbContext _context;
18
            0 references
19          public TodosController(ApplicationDbContext context)
20          {
21              _context = context;
22          }
23
24          // GET: api/Todos
25          [HttpGet]
            0 references
26          public async Task<ActionResult<IEnumerable<Todo>>> GetTodos()
27          {
28              return await _context.Todos.ToListAsync();
29          }
30
31          // GET: api/Todos/5
32          [HttpGet("{id}")]
            0 references
33          public async Task<ActionResult<Todo>> GetTodo(Guid id)
34          {
35              var todo = await _context.Todos.FindAsync(id);
36
37              if (todo == null)
38              {
39                  return NotFound();
40              }
41
42              return todo;
43          }
44
45          // PUT: api/Todos/5
46          // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
47          [HttpPut("{id}")]
            0 references
48          public async Task<IActionResult> PutTodo(Guid id, Todo todo)
49          {
```

API Controller with CRUD actions is generated and ready to use.

JSON

Steps to install a JSON library for data serialization and deserialization:

- Right click on the Client Project > Manage NuGet Packages
- Search for “Newtonsoft.JSON”. It’s the most popular package on Nuget and it will show up at the top without even searching for it
- Click install



Make HTTP requests against our API

Add API calls in Index, Create and Edit pages of the Todo entity.

Index page

Let's fetch all todo items from the API.

```
@code {
    private List<Todo> Todos = new List<Todo>();

    protected override async Task OnInitializedAsync()
    {
        try
        {
            Todos = await Http.GetFromJsonAsync<List<Todo>>("/api/Todos");
        }
        catch (AccessTokenNotAvailableException exception)
        {
            exception.Redirect();
        }
        catch (Exception e)
        {

        }
    }
}
```

Create page

Make a post requests for creating a new todo item.

```
@code {
    private Todo Todo { get; set; } = new Todo();

    private async void HandleValidSubmit()
    {
        try
        {
            var response = await Http.PostAsJsonAsync("/api/Todos", Todo);
            response.EnsureSuccessStatusCode();

            var content = await response.Content.ReadAsStringAsync();
            var todo = JsonConvert.DeserializeObject<Todo>(content);
            Navigation.NavigateTo($"todos/edit/{todo.Id}");
        }
        catch (AccessTokenNotAvailableException exception)
        {
            exception.Redirect();
        }
        catch (Exception e)
        {

        }
    }
}
```

Edit page

Fetch data for a specified Todo item.

```
@code {
    [Parameter] public Guid Id { get; set; }
    private Todo Todo { get; set; } = new Todo();

    protected override async Task OnInitializedAsync()
    {
        try
        {
            Todo = await Http.GetFromJsonAsync<Todo>($"/api/Todos/{Id}");
        }
        catch (AccessTokenNotAvailableException exception)
        {
            exception.Redirect();
        }
        catch (Exception e)
        {

        }
    }
```

Using Put method to update the item.

```
private async void HandleValidSubmit()
{
    try
    {
        var response = await Http.PutAsJsonAsync($"/api/Todos/{Todo.Id}", Todo);
        response.EnsureSuccessStatusCode();
    }
    catch (AccessTokenNotAvailableException exception)
    {
        exception.Redirect();
    }
    catch (Exception e)
    {

    }
}
```

Delete page

Try to do it yourself. **Hint:** It's exactly the same as GET /api/Todos/{id}, the only difference is that we are using DELETE method.

Authorization

If token is expired, user is redirected to the login page.

```
@code {
    private List<Todo> Todos = new List<Todo>();

    protected override async Task OnInitializedAsync()
    {
        try
        {
            Todos = await Http.GetFromJsonAsync<List<Todo>>("/api/Todos");
        }
        catch (AccessTokenNotAvailableException exception)
        {
            exception.Redirect();
        }
        catch (Exception e)
        {

        }
    }
}
```

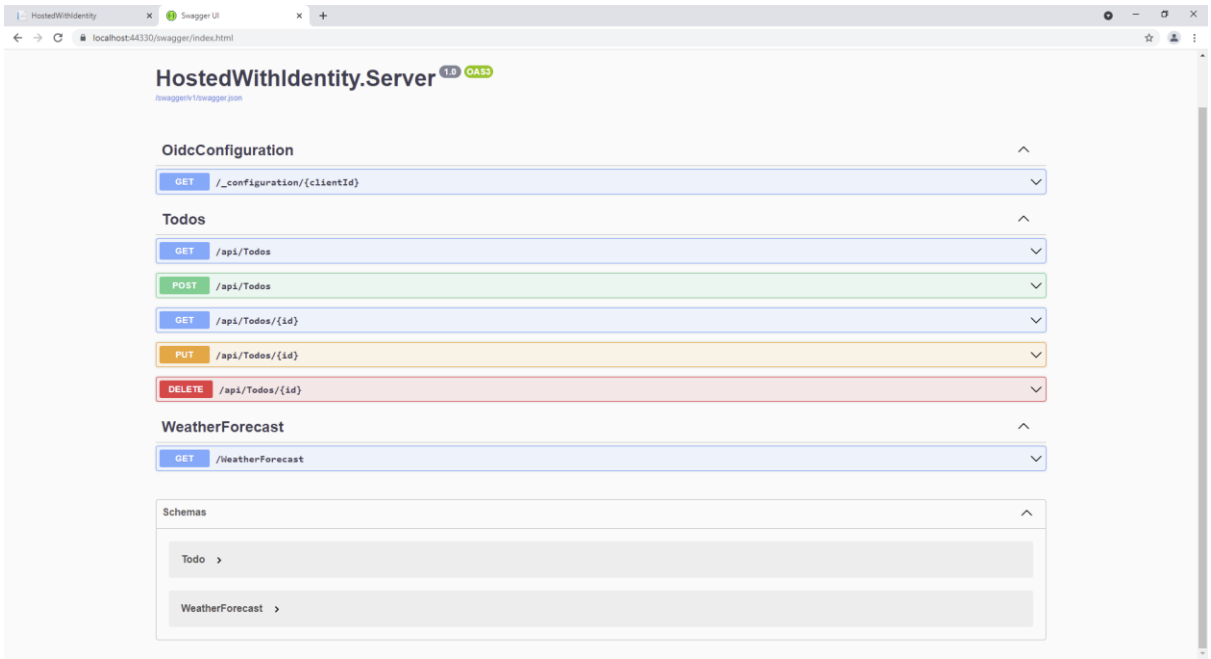
[Authorize] attribute in Index, Create and Edit pages.

```
1   @page "/todos"
2
3   @attribute [Authorize]
4   @inject HttpClient Http
5
6   <h1>Todos index</h1>
7
8   <div class="table-responsive">
```

[Authorize] attribute in API TodosController.

```
namespace HostedWithIdentity.Server.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class TodosController : ControllerBase
    {
        private readonly ApplicationDbContext _context;
```

Swagger



Steps to set it up:

- Install NuGet package Swashbuckle.AspNetCore in the Server project
- Add services.AddSwaggerGen(); to ConfigureServices method in Startup.cs
- Add app.UseSwagger(); and app.UseSwaggerUI(); to Configure method in Startup.cs

To open Swagger type in your browser: {host:port}/swagger.

Useful links

- https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-5.0&tabs=visual-studio