Building Angular Applications using ASP.NET Core Angular CLI Template

Posted by: Ravi Kiran (../../Author.aspx?AuthorName=Ravi Kiran) , on 4/4/2018, in **Category** ASP.NET Core (../../BrowseArticles.aspx?CatID=88)

Abstract: ASP.NET core now has good support for Angular development using Angular CLI. In this tutorial, build a simple CRUD application using the ASP.NET Core Angular CLI template.

Microsoft built ASP.NET core to make modern application development easier and efficient. ASP.NET core makes it possible to build and deploy .NET based server web applications across any of the major platforms available.

On the other hand, front-end web is evolving very rapidly. The tooling to build front-end applications is getting matured to meet the developers' needs. This brings up a challenge to the server platforms to support the front-end ecosystem inside their boundaries.

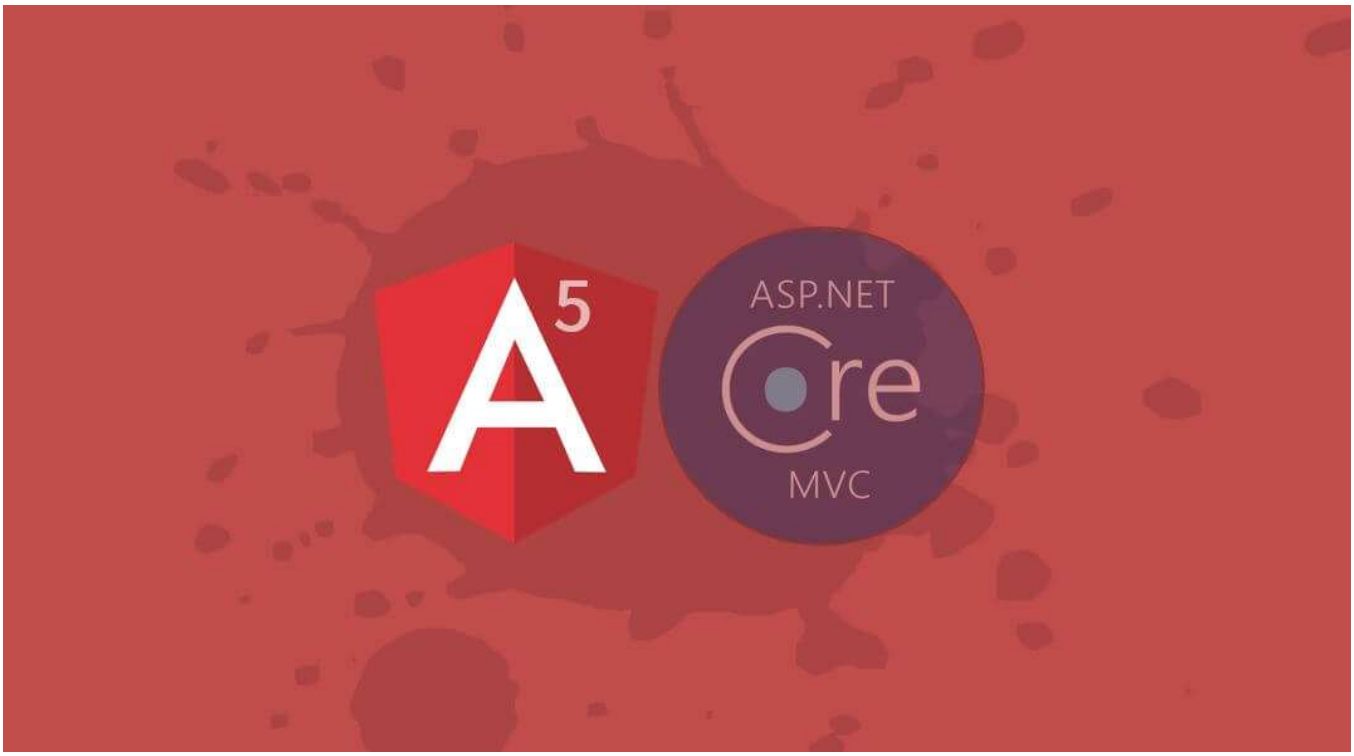
The ASP.NET team has built Angular and React templates to support development of front-end based applications with ASP.NET core as the backend.

The Angular team built Angular CLI (https://www.dotnetcurry.com/angularjs/1409/angular-cli-tutorial) to address the difficulties that developers face while setting up the environment to build a sophisticated web application. It reduces the time and effort required to generate a new Angular application. Using the commands it provides, one can generate an application, generate different code blocks of Angular, add unit test spec files, run tests, run the applications in development mode and generate build files to deploy the application.

These features are hard to recreate. Anyone would love to have these features while working on Angular.

So, the ASP.NET team released a new template by including Angular CLI (https://www.dotnetcurry.com/angularjs/1409/angular-cli-tutorial). This template is still in RC at the time of this writing and is expected to be ready for production in a few months.

This tutorial will show you how to create a simple CRUD based application using the ASP.NET core Angular template.



Angular App with ASP.NET Core Angular CLI Template - System Setup

To follow along with this article and build the sample, your system should have the following software installed:

- Visual Studio 2017 (https://www.visualstudio.com/downloads/)

- .NET core 2.0 or above (https://www.microsoft.com/net/learn/get-started/windows)

- Node.js (https://nodejs.org/en/)

- Angular CLI: To be installed as a global npm package using the following command

```
> npm install -g @angular/cli@latest
```

Angular CLI is installed using npm, so the system should have Node.js installed before the npm command is executed.

The ASP.NET core Angular template to be used in this article is still a release candidate (RC) and it is not available on Visual Studio's template dialog box. The template has to be installed using the following command:

```
> dotnet new --install Microsoft.DotNet.Web.Spa.ProjectTemplates::2.0.0-rc1-final
```

Once these installations are complete, you can start using the new template to build applications.

Understanding the Application Structure

We are going to build a video tracker application. Using this application, one can enter his or her favorite set of videos with their links and can watch the video by clicking the link whenever he or she wants to watch it.

At first, we need to get this project generated using the template. Open a command prompt, go to the folder where you want to place your code and run the following command:

```
> dotnet new angular -o VideoTracker
```

This command will generate the web application containing ASP.NET core and Angular. The code gets added to a new folder named *VideoTracker.* This folder contains the *VideoTracker.csproj* file, open the project in Visual Studio 2017 using this file.

The folder structure is shown in Figure 1.

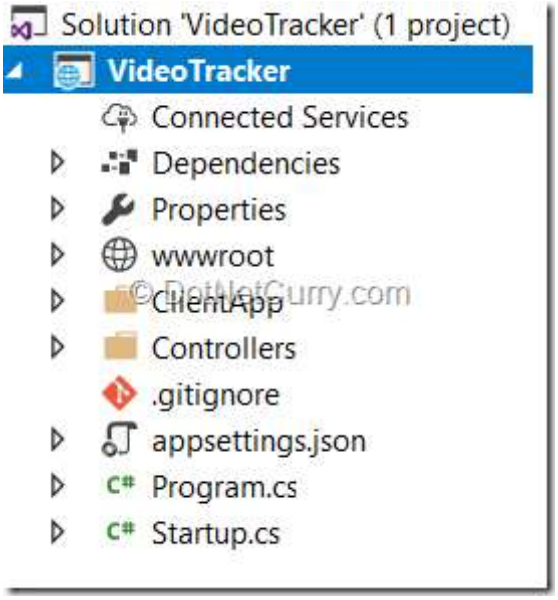


Figure 1 – Folder Structure

The following listing describes the files and folders in the above screenshot:

- *ClientApp* folder contains the Angular CLI based front end application
- *Controllers* folder contains a sample controller with an API endpoint
- The file *appsettings.json* contains some settings used by the ASP.NET core application
- The *Program.cs* file has the *Main* method to start the application
- The *Startup.cs* file sets up the services and middlewares for the ASP.NET core application

The following screenshot shows structure of the *ClientApp* folder:

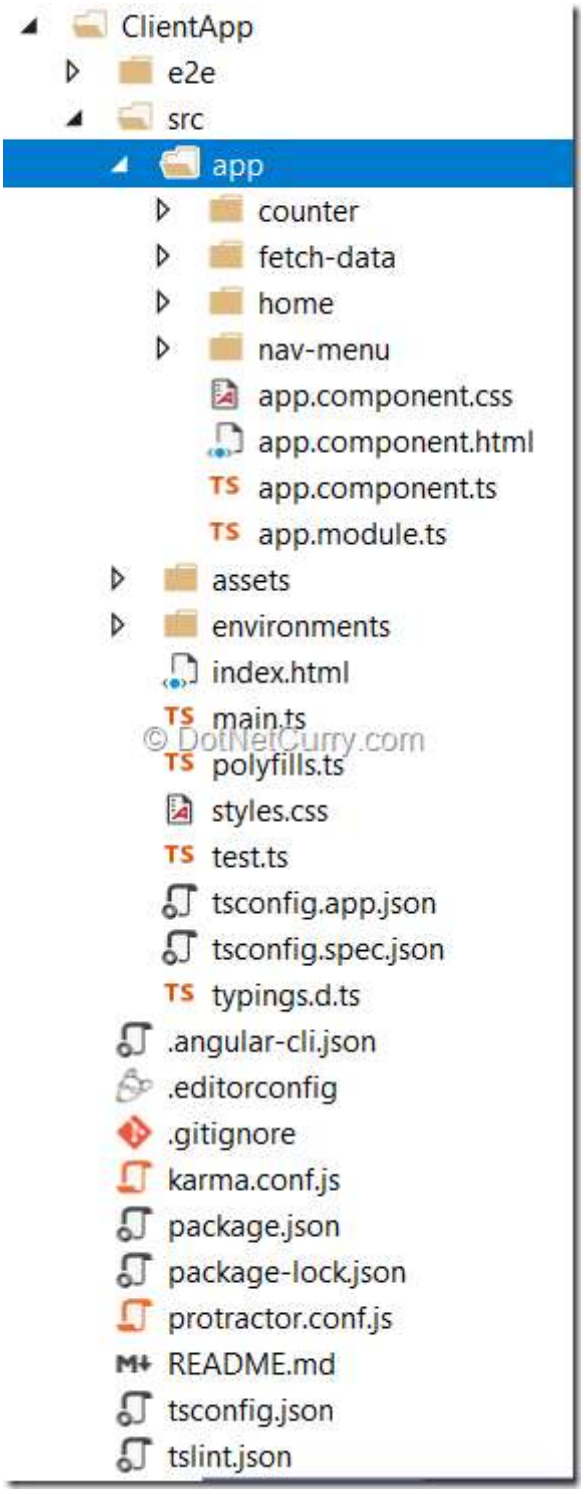


Figure 2 – Structure of ClientApp folder

If you are familiar with the files generated by Angular CLI, most of the files in Figure 2 would look familiar to you. If not, read the Angular CLI introduction (https://www.dotnetcurry.com/angularjs/1409/angular-cli-tutorial) article to understand the structure.

The *src* folder contains the front-end code that gets rendered on the browser. To build the Video Tracker application, we will modify the files inside the *src* folder.

The following listing describes the vital parts of the *src* folder:

- The file *main.ts* bootstraps the application

- The file *app.module.ts* defines the Angular module of the application and configures the routes. The following snippet shows the modules imported and the route configuration added to *AppModule:*

```
imports: [
  BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
  HttpClientModule,
  FormsModule,
  RouterModule.forRoot([
    { path: '', component: HomeComponent, pathMatch: 'full' },
    { path: 'counter', component: CounterComponent },
    { path: 'fetch-data', component: FetchDataComponent },
  ])
]
```

Notice the *BrowserModule* in the imports array. Call to the method *withServerTransition* is made to support server-side rendering of the Angular application. The application has three routes configured.

- The *AppComponent* in the file *app.component.ts* contains the *router-outlet* component, which is the placeholder to display the routes. It uses the *NavMenuComponent* to show the menu items that help to navigate between the routes.

- The components supplied to the routes carry out simple tasks. The *HomeComponent* has a static HTML template displaying some information about ASP.NET core, Angular, Bootstrap and the template. The *CounterComponent* has a basic data binding example. The *FetchDataComponent* makes a call to the sample API added in the project and displays the result in a table.

To run any Angular CLI commands to generate the new angular code files, you can open a command prompt in the *ClientApp* folder and run the command there. This will be covered later in this article.

The *Startup.cs* file contains the configuration to start the client application from the *ClientApp* folder. To run the application in development mode, it uses the “npm start” script configured in the client application. The “npm start” script uses the “ng serve” command in turn to start the Angular application in development mode. The following snippet in the *Configure* method does this:

```
app.UseSpa(spa =>
{
    // To learn more about options for serving an Angular SPA from ASP.NET Core,
    // see https://go.microsoft.com/fwlink/?linkid=864501

    spa.Options.SourcePath = "ClientApp";

    if (env.IsDevelopment())
    {
        spa.UseAngularCliServer(npmScript: "start");
    }
});
```

While running the application in production mode, it takes the static files from *ClientApp/dist* folder. This is because the Angular CLI spits out the bundled static files inside this folder. The following snippet in the *ConfigureServices* method of the *Startup* class does this:

```
// In production, the Angular files will be served from this directory
services.AddSpaStaticFiles(configuration =>
{
    configuration.RootPath = "ClientApp/dist";
});
```

Build the project using Build menu or by pressing Ctrl+Shift+B. This will take a couple of minutes, as it is going to install the npm packages added to the Angular application. Once the build completes, run it using Ctrl+F5. You will see a bootstrap based Angular application loaded in the browser.

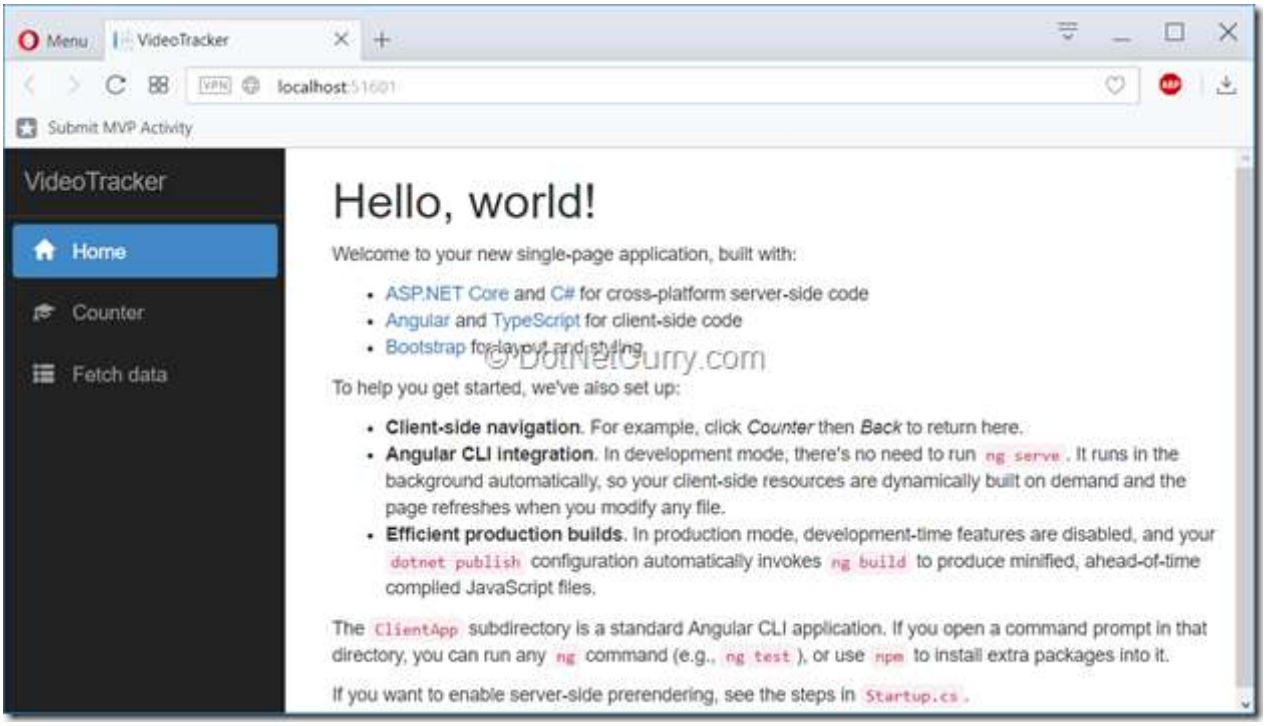


Figure 3 – Running instance of generated application

As you see, the generated application has three pages. They are registered as Angular routes. The Counter page shows how data binding works in Angular with a number incrementing on click of a button. And the Fetch Data page calls the sample API in the application to fetch some data and displays it in a table.

Adding Database and Building API

Now that we understood the structure and the way the application works, let's modify it to build the video tracker. Let's get the database and the API ready to serve the data. We will create the database using Entity Framework Core.

Add a folder named *Model* to the project. Add a new C# class file to this folder and name it *Video.cs.* Change contents of the file as follows:

```
namespace VideoTracker.Model
{
    public class Video
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Link { get; set; }
        public bool Watched { get; set; }
        public string Genre { get; set; }
    }
}
```

Add another file to the *Model* folder and name it *VideoTrackerContext.* This class will be the DbContext class for the database we are going to create. Open this file and replace it with the following code:

```
using Microsoft.EntityFrameworkCore;

namespace VideoTracker.Model
{
    public class VideoTrackerContext: DbContext
    {
        public VideoTrackerContext(DbContextOptions<VideoTrackerContext> options)
            :base(options)
        {
        }

        public DbSet<Video> Videos { get; set; }
    }
}
```

Open the file *appsettings.json* and add the following connection string to it:

```
"ConnectionStrings": {
  "VideoTrackerContext": "Server=
(localdb)\\mssqllocaldb;Database=VideoTrackerDb;Trusted_Connection=True;MultipleActiveRe
}
```

The `DbContext` class has to be registered in the services of the application. To do so, add the following snippet to the *ConfigureServices* method of the *Startup* class:

```
services.AddDbContext<VideoTrackerContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("VideoTrackerContext")));
```

Now we have the connection string, model class and the context class ready.

To create the database and seed some data into it, we need to add EF Core migrations. Migrations need a context factory class. Add a new C# class to the *Model* folder and name it *VideoContextFactory.cs.* Place the following code inside this file:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.Configuration;
using System.IO;

namespace VideoTracker.Model
{
    public class VideoContextFactory : IDesignTimeDbContextFactory<VideoTrackerContext>
    {
        public VideoTrackerContext CreateDbContext(string[] args)
        {
            IConfigurationRoot configuration = new ConfigurationBuilder()
                            .SetBasePath(Directory.GetCurrentDirectory())
                            .AddJsonFile("appsettings.json")
                            .Build();
            var optionsBuilder = new DbContextOptionsBuilder<VideoTrackerContext>();
            optionsBuilder.UseSqlServer(configuration.GetConnectionString("VideoTracker(

            return new VideoTrackerContext(optionsBuilder.Options);
        }
    }
}
```

To access the database created, we need *Microsoft.EntityFrameworkCore.Tools* package and need to run migration commands in the package manager console. Open the Package Manager Console in Visual Studio and run the following commands in the given order:

```
> Install-Package Microsoft.EntityFrameworkCore.Tools
> Add-Migration Initial
> Update-Database
```

Once these commands are successful, you will have access to the database and the table created. You can see the database in the SQL Server Object Explorer. It will have the Videos table created with the five columns as specified in the *Video* class earlier.

Let's add some data to this table, so that we can see the data in response to the APIs.

Add a new file to the *Model* folder and name it *Seed.cs.* Add the following code to it:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Linq;

namespace VideoTracker.Model
{
    public class Seed
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new VideoTrackerContext(
                serviceProvider.GetRequiredService<DbContextOptions<VideoTrackerContext>
()))
            {
                // Look for any movies.
                if (context.Videos.Any())
                {
                    return;   // DB has been seeded
                }

                context.Videos.AddRange(
                    new Video
                    {
                        Title = "Raghuvamsa Sudha",
                        Link = "https://www.youtube.com/watch?v=l89ZaBqYiTA",
                        Genre = "Music",
                        Watched= true
                    },
                    new Video
                    {
                        Title = "Kapil finds a match for Sarla",
                        Link = "https://www.youtube.com/watch?v=GfHEPKgkkpw",
                        Genre = "Comedy",
                        Watched = true
                    },
                    new Video
                    {
                        Title = "Arvind Gupta TED Talk",
                        Link = "https://www.youtube.com/watch?v=KnCqR2yUXoU",
                        Genre = "Inspirational",
                        Watched = true
                    },
                    new Video
                    {
                        Title = "Event Loop - Philip Roberts",
                        Link = "https://www.youtube.com/watch?v=8aGhZQkoFbQ",
                        Genre = "Tech",
                        Watched = true
                    }
                );
                context.SaveChanges();
            }
        }
    }
}
```

The *Initialize* method of the *Seed* class has to be called as soon as the application starts. Modify the *Main* method in the *Program.cs* file as follows:

```
public static void Main(string[] args)
{
    // Creaying a host based on the Startup class
    var host = CreateWebHostBuilder(args).Build();

    // Creating a scope for the application
    using (var scope = host.Services.CreateScope())
    {
        // Getting service provider to resolve dependencies
        var services = scope.ServiceProvider;

        try
        {
            // Initializing the database
            Seed.Initialize(services);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred seeding the DB.");
        }
    }

    host.Run();
}
```

If you run the application after saving the above changes, the data would be inserted to the Videos table.

As the database is ready with some data, let's add the API. Right click on the *Controllers* folder and choose Add > Controller. From the menu, choose API Controller with actions using Entity Framework.

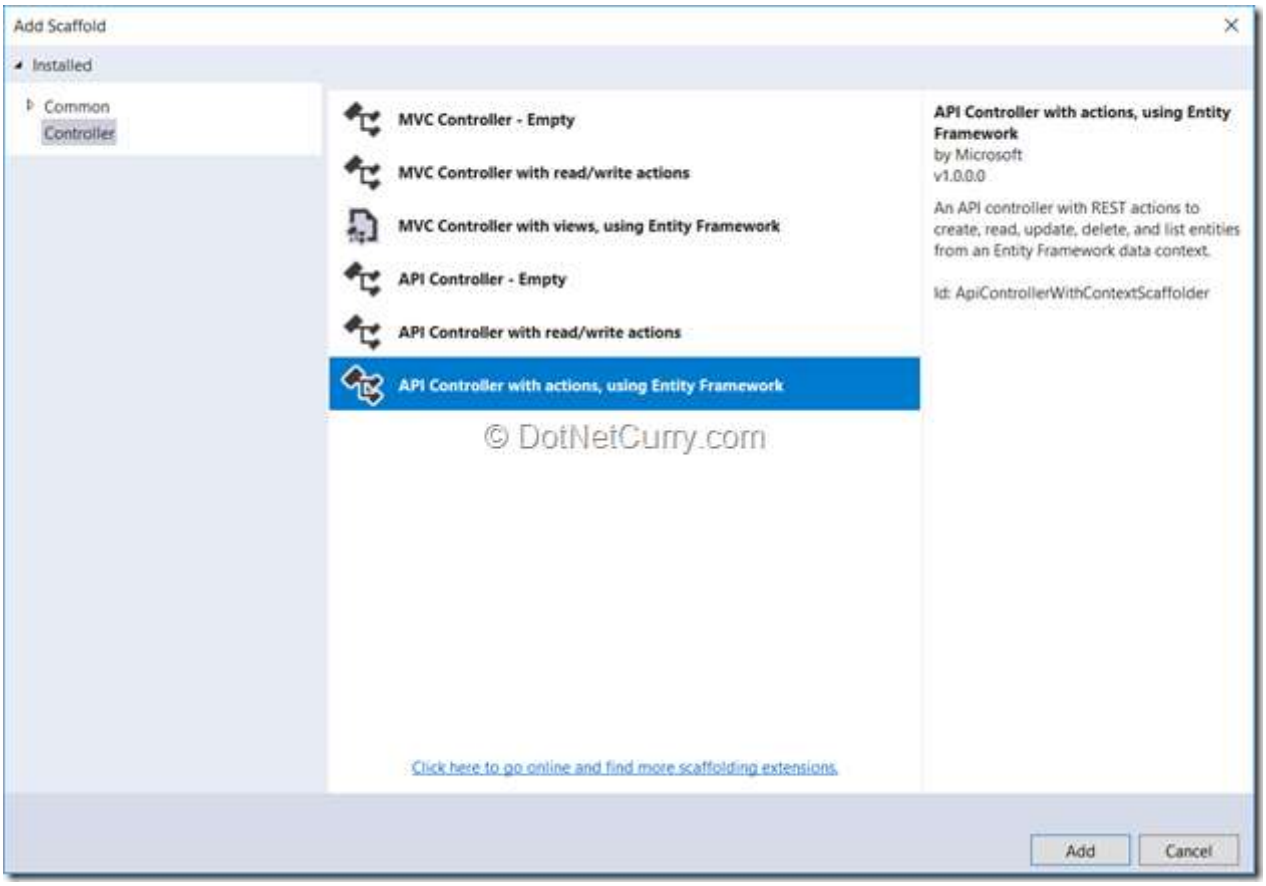


Figure 4 – Add control dialog

Click on the add button after choosing the option. This will show another prompt asking for the details of the model and the context class. Change the values as follows:

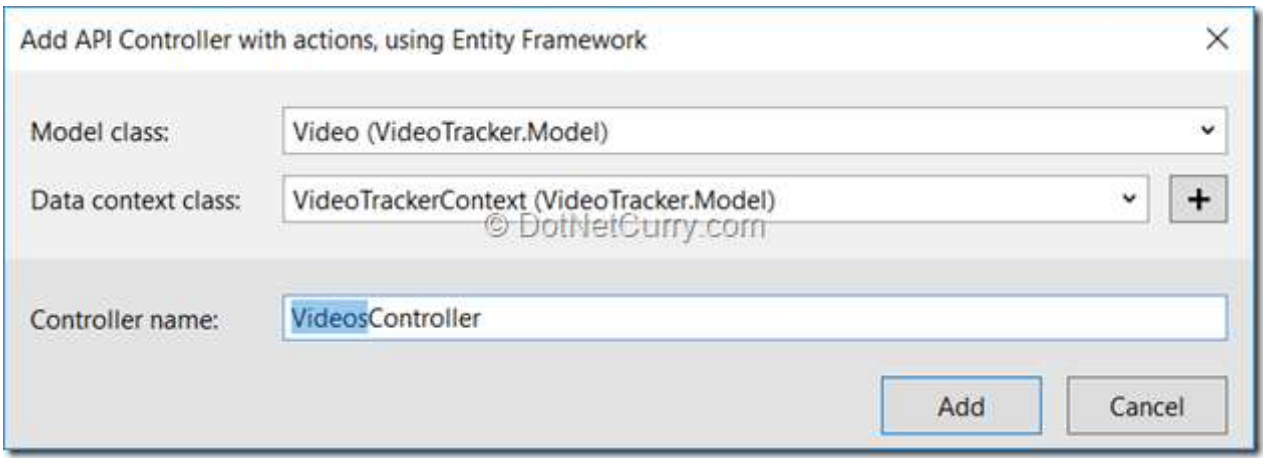


Figure 5 – Controller with actions dialog

Once you hit the add button, it will create the *ValuesController* performing CRUD operations on the *Videos* table. We don't need to make any changes to this file, we can use it as is. Run the application and change the URL in the browser to http://localhost:<port-no>/api/Videos. This will display the set of videos added to the table. See Figure 6.


```
(4) [
- {
    id: 1,
    title: "Raghuvamsa Sudha",
    link: https://www.youtube.com/watch?v=189ZaBqYiTA,
    watched: true,
    genre: "Music"
  },
- {
    id: 2,
    title: "Kapil finds a match for Sarla",
    link: https://www.youtube.com/watch?v=GfHEPKgkkpw,
    watched: false,
    genre: "Comedy"
  },
- {
    id: 3,
    title: "Arvind Gupta TED Talk",
    link: https://www.youtube.com/watch?v=KnCqR2yUXoU,
    watched: false,
    genre: "Inspirational"
  },
- {
    id: 4,
    title: "Event Loop - Philip Roberts",
    link: https://www.youtube.com/watch?v=8aGhZQkoFbQ,
    watched: false,
    genre: "Tech"
  },
]
```


Figure 6 – Output of the videos API

Building Video Tracker Client App

As stated earlier, we need to modify the *ClientApp/src* folder to build the client part of the application. The application needs two views, one to display the list of videos and the other to add a new video. The Angular components required for these views can be generated using the Angular CLI commands. Open a command prompt and move to the *ClientApp* folder. Run the following commands to get the components generated:

```
> ng generate component video-list
> ng generate component add-video
```

Angular CLI provides shorter versions of these commands to ease the usage. The command to generate the *video-list* component can be replaced with the following:

```
> ng g c video-list
```

These commands add components with their basic skeleton and register them in the *AppModule*. These components have to be registered as routes, so that we can access them as Angular views.

As we will be using these components going forward, it is better to remove the components that were added by the template. Open the file *app.module.ts* and change the code in the file as shown in the following snippet:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, InjectionToken } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { VideoListComponent } from './video-list/video-list.component';
import { AddVideoComponent } from './add-video/add-video.component';

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    VideoListComponent,
    AddVideoComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot([
      { path: '', component: VideoListComponent, pathMatch: 'full' },
      { path: 'addvideo', component: AddVideoComponent }
    ])
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now this file is free from the components added by the template and it registers the new components in the routes.

The navigation menu still points to the old routes. Let's change it to use the new routes there. Open the file *nav-menu.component.html* and change the code of this file as follows:

```
<div class='main-nav'>
    <div class='navbar navbar-inverse'>
        <div class='navbar-header'>
            <button type='button' class='navbar-toggle' data-toggle='collapse' data-
target='.navbar-collapse' [attr.aria-expanded]='isExpanded' (click)='toggle()'>
                <span class='sr-only'>Toggle navigation</span>
                <span class='icon-bar'></span>
                <span class='icon-bar'></span>
                <span class='icon-bar'></span>
            </button>
            <a class='navbar-brand' [routerLink]='["/"]'>VideoTracker</a>
        </div>
        <div class='clearfix'></div>
        <div class='navbar-collapse collapse' [ngClass]='{ "in": isExpanded }'>
            <ul class='nav navbar-nav'>
                <li [routerLinkActive]='["link-active"]' [routerLinkActiveOptions]='{
exact: true }'>
                    <a [routerLink]='["/"]' (click)='collapse()'>
                        <span class='glyphicon glyphicon-home'></span> Video List
                    </a>
                </li>
                <li [routerLinkActive]='["link-active"]'>
                    <a [routerLink]='["/addvideo"]' (click)='collapse()'>
                        <span class='glyphicon glyphicon-plus'></span> Add Video
                    </a>
                </li>
            </ul>
        </div>
    </div>
</div>
```

Save these changes and run the application. You will see the new menu items in the menu bar as shown in Figure 7:

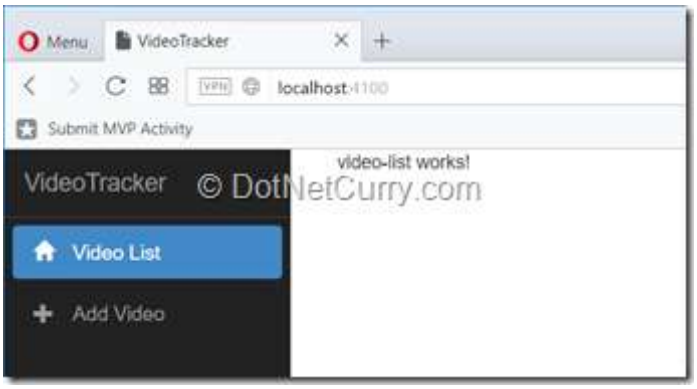


Figure 7 – Application with new menu items

Before modifying the components to show the new views, let's create a service to interact with the backend. The reason behind creating a service to perform the data operations is to keep the components focused on building data required for its view alone. Also, as the service would be available to all the components in the application, anyone can use it without having to repeat the logic. Run the following command to generate the service:

```
> ng g s videos -m app.module
```

This command adds the *VideosService* and registers in the *AppModule.* This service will interact with the web APIs created in the *VideosController.* Open the newly added file *videos.service.ts* and modify the code in this file as shown below:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from "rxjs/Observable";
import { Video } from "./video";

@Injectable()
export class VideosService {

  constructor(private httpClient: HttpClient) { }

  // Getting the list of videos
  getVideos(): Observable<Video[]> {
    return this.httpClient.get<Video[]>('/api/Videos');
  }

  // Adding a new video by calling post
  addVideo(video: Video) {
    return this.httpClient.post('/api/Videos',video);
  }

  // Marking a video as watched using the PUT API
  setWatched(video: Video) {
    video.watched = true;

    return this.httpClient.put(`/api/Videos/${video.id}`, video);
  }
}
```

The *videos-list* component will use the *getVideos* method to fetch the videos and show them in a table. It needs to update the *watched* property of the video when link of a video is clicked. It is better to write a method in the component to perform both these tasks. To open the link programmatically, the *window* object has to be used.

The application will be hard to test if the *window* object is used from global scope. Even if we are not going to use unit testing in this article, it is always good to use a better way to access the global objects. For this, we need to create a token using *InjectionToken* to register the object and inject it using the token. Add a new file to the *src* folder and name it *window.ts*. Add the following code to it:

```
import { InjectionToken } from '@angular/core';

const WINDOW = new InjectionToken<any>('window');

export const windowProvider = { provide: WINDOW, useValue: window };
```

The object *windowProvider* can be used to register the injectable. Open the file *app.module.ts* and import the *windowProvider* object.

```
import { windowProvider } from './window';
```

Modify the *providers* array of the module metadata as shown in the following snippet to register the token:

```
providers: [
  VideosService,
  windowProvider
]
```

Now we have all the objects ready to be used in the components. Open the file *video-list.component.ts* and modify the code as shown below:

```
import { Component, OnInit, Inject } from '@angular/core';

import { VideosService } from '../videos.service';
import { Video } from "../video";
import { windowProvider } from '../window';

@Component({
  selector: 'app-video-list',
  templateUrl: './video-list.component.html',
  styleUrls: ['./video-list.component.css']
})
export class VideoListComponent implements OnInit {
  public videos: Video[];

  constructor(private videosService: VideosService,
    @Inject(windowProvider.provide) private window: Window) { }

  ngOnInit() {
    this.videosService.getVideos()
      .subscribe(videos => {
        this.videos = videos;
     });
  }

  watch(video: Video) {
    this.videosService.setWatched(video)
      .subscribe(v => console.log(v));
    this.window.open(video.link, '_blank');
  }

}
```

Notice the way the *window* object is injected in the above component. It Is injected using the *Inject* decorator and the token is passed into the decorator.

The template has to be modified to display the data in the table. Place the following code in the file *video-list.component.html:*

```
<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Genre</th>
      <th>Watched</th>
      <th>Watch</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let video of videos">
      <td>{{video.title}}</td>
      <td>{{video.genre}}</td>
      <td>
        <span *ngIf="video.watched">Yes</span>
        <span *ngIf="!video.watched">No</span>
      </td>
      <td>
        <a href="{{video.link}}" target="_blank" (click)="watch(video)">Watch
Video</a>
      </td>
    </tr>
  </tbody>
</table>
```

Save these files and run the application. Figure 8 shows the home page displaying a list of videos.

Figure 8 – Video tracker with videos list

Let's get the *add-video* component ready to add new videos to the application. Open the file *add-video.component.ts* and replace it with the following code:

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

import { VideosService } from '../videos.service';
import { Video } from "../video";

@Component({
  selector: 'app-add-video',
  templateUrl: './add-video.component.html',
  styleUrls: ['./add-video.component.css']
})
export class AddVideoComponent {
  public genres: string[];
  public video: Video = {};

  constructor(
    private router: Router,
    private videosSvc: VideosService) {
    this.genres = [
      "",
      "Music",
      "Comedy",
      "Inspirational",
      "Tech",
      "News",
      "Sports"
    ];
  }

  addVideo() {
    this.videosSvc.addVideo(this.video)
      // The subscribe method here adds a listener to the asynchronous add
      // operation and the listener is called once the video is added.
      // Here, we are switching to the video list page after adding the video
      .subscribe(() => {
        this.router.navigateByUrl('/');
      });
  }
}
```

The last thing to do is to modify the template of the *add-video* component to add a form. The form will have basic required field validations to ensure that the fields have values when the form is submitted.

Place the following code in the file *add-video.component.html:*

```
<div>
  <form #videoForm="ngForm" (ngSubmit)="addVideo()">
    <div class="form-group">
      <label for="name">Title</label>
      <input type="text" class="form-control"
             name="title"
             required
             [(ngModel)]="video.title"
             #title="ngModel">
    </div>

    <div class="form-group">
      <label for="link">Link</label>
      <input type="text" class="form-control"
             name="link"
             required
             [(ngModel)]="video.link"
             #link="ngModel">
    </div>

    <div class="form-group">
      <label for="genre">Genre</label>
      <select class="form-control"
              name="genre"
              required
              [(ngModel)]="video.genre"
              #genre="ngModel">
        <option *ngFor="let genre of genres" [value]="genre">{{genre}}</option>
      </select>
    </div>

    <button type="submit" class="btn btn-success"
            [disabled]="!videoForm.form.valid">Submit</button>
    <button type="reset" class="btn"
            (click)="videoForm.reset()">Reset</button>
  </form>
</div>
```

Now the application is complete. Run it and you can add a new video. And then you can see it in the video list page.

Conclusion

As we saw, ASP.NET core now has good support for Angular development using Angular CLI. The template doesn't try to hide any of the features of Angular CLI and provides a good support for building and deploying Angular applications with ASP.NET core. We can use this template to build and deploy dynamic web applications using the goodness of both the technologies.

Download the entire source code *from GitHub at bit.ly/dncm35-aspcorecli* (http://bit.ly/dncm35-aspcorecli)

This article was technically reviewed by Keerti Kotaru (https://www.dotnetcurry.com/author/v-keerti-kotaru).

WHAT OTHERS ARE READING!

Facade Design Pattern: Still relevant in ASP.NET Core? (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1468)

End-to-End (E2E) Testing of ASP.NET Core Applications using Selenium and Nightwatch.js (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1433)

Dependency Injection in ASP.NET Core - Demystified (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1426)

Integration Testing for ASP.NET Core Applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1420)

Unit Testing ASP.NET Core Applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1414)

Using Azure database for PostgreSQL in ASP.NET Core (with EF Core) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1410)

Was this article worth reading? Share it with fellow developers too. Thanks!

Share on Facebook (https://facebook.com/sharer/sharer.php?u=https%3A%2F%2Fwww.dotnetcurry.com%2Fangularjs%2F1432%2Faspnet-core-apps-using-angular-cli-template) Share on Twitter (https://twitter.com/intent/tweet/?text=Building%20Angular%20Applications%20using%20ASP.NET%20Core%20Angular%20CLI%20Template%20%7C%20DotNetCurry&url=https%3A%2F%2Fwww.dotnetcurry.com%2Fangularjs%2F1432%2Faspn

et-core-apps-using-angular-cli-template) Share on LinkedIn (https://www.linkedin.com/shareArticle?mini=true&url=https%3A%2F%2Fwww.dotnetcurry.com%2Fangularjs%2F1432%2Faspnet-core-apps-using-angular-cli-template&title=Building%20Angular%20Applications%20using%20ASP.NET%20Core%20Angular%20CLI%20Template%20%7C%20DotNetCurry) Share on Google+ (https://plus.google.com/share?url=https%3A%2F%2Fwww.dotnetcurry.com%2Fangularjs%2F1432%2Faspnet-core-apps-using-angular-cli-template)

AUTHOR

Rabi Kiran (a.k.a. Ravi Kiran) is a developer working on Microsoft Technologies at Hyderabad. These days, he is spending his time on JavaScript frameworks like AngularJS, latest updates to JavaScript in ES6 and ES7, Web Components, Node.js and also on several Microsoft technologies including ASP.NET 5, SignalR and C#. He is an active blogger (http://sravi-kiran.blogspot.com/), an author at SitePoint (http://www.sitepoint.com/author/rkiran/) and at DotNetCurry (http://www.dotnetcurry.com/author/ravi-kiran). He is rewarded with Microsoft MVP (Visual Studio and Dev Tools) and DZone MVB awards for his contribution to the community

FEEDBACK - LEAVE US SOME ADULATION, CRITICISM AND EVERYTHING IN BETWEEN!

Click here to post your Comments

CATEGORIES

- .NET Web
- .NET Framework, Visual Studio and C#
- Patterns & Practices
- Cloud and Mobile
- JavaScript
- .NET Desktop
- Interview Questions & Product Reviews

JOIN OUR COMMUNITY

51,059
fans
(https://www.facebook.com/dotnetcurry)

7,564
followers
(https://www.twitter.com/dotnetcurry)

102,312
subscribers
(https://www.dotnetcurry.com/magazine/)

POPULAR ARTICLES

Writing Honest Methods in C# (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1449)

.NET Core Application Development in Visual Studio Code (VS Code) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1451)

C# Sharding and Multithreading - Deep Dive (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1457)

Angular Evolution - Version 1.x to 6 (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1453)

Command Query Separation (CQS) - A simple but powerful pattern (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1461)

Interview with the C# Boss - Mads Torgersen (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1455)

Xamarin.Forms 3 - Cheat sheet (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1452)

Writing Pure Code in C# (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1464)

Blazor - .NET in the browser (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1460)

The Evolution of C# (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1465)

Angular HttpClient Deep Dive (Headers, HTTP events, non-JSON data and Interceptors) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1448)

How to Choose the Right .NET Collection Class? (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1466)

Integration Testing made Simple for CRUD applications with SqlLocalDB (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1456)

Ahead of Time Compilation - AoT in Angular (Performance Improvements) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1450)

Serverless Apps with Azure Functions (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1459)

C# .NET BOOK

(http://www.dotnetcurry.org/r/dnc-csharpbk-web-300x600x2-green)

Tags

ASP.NET MVC (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET-MVC)

ASP.NET CORE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET-CORE)

ASP.NET (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET)

SHAREPOINT (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/SHAREPOINT)

DESIGN PATTERNS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/PATTERNS-PRACTICES)

C# (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/CSHARP)

LINQ (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/LINQ)

WPF (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WPF)

WCF (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WCF)

VISUAL STUDIO (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/VISUALSTUDIO)

VSTS & TFS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/VSTS-TFS)

AZURE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WINDOWS-AZURE)

ENTITY FRAMEWORK (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ENTITYFRAMEWORK)

ANGULAR.JS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ANGULARJS)

REACT.JS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/REACTJS)

JQUERY (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/JQUERY-ASPNET)

JAVASCRIPT (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/HTML5-JAVASCRIPT)

HTML5 (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/HTML5-JAVASCRIPT)

.NET CORE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/DOTNET-STANDARD-CORE)

.NET FRAMEWORK (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/DOTNETFRAMEWORK)

(https://www.dotnetcurry.com)

SERVER-SIDE

ASP.NET (https://www.dotnetcurry.com/tutorials/aspnet)

ASP.NET Core (https://www.dotnetcurry.com/tutorials/aspnet-core)

ASP.NET MVC (https://www.dotnetcurry.com/tutorials/aspnet-mvc)

WCF (https://www.dotnetcurry.com/tutorials/wcf)

SharePoint (https://www.dotnetcurry.com/tutorials/sharepoint)

CLIENT-SIDE

Angular.js (https://www.dotnetcurry.com/tutorials/angularjs)

React.js (https://www.dotnetcurry.com/tutorials/reactjs)

jQuery (https://www.dotnetcurry.com/tutorials/jquery-aspnet)

Backbone.js (https://www.dotnetcurry.com/tutorials/backbonejs)

HTML5 (https://www.dotnetcurry.com/tutorials/html5-javascript)

CSS (https://www.dotnetcurry.com/tutorials/bootstrap-css)

.NET

C# (https://www.dotnetcurry.com/tutorials/csharp)

Visual Studio (https://www.dotnetcurry.com/tutorials/visualstudio)

VSTS & TFS (https://www.dotnetcurry.com/tutorials/vsts-tfs)

LINQ (https://www.dotnetcurry.com/tutorials/linq)

Entity Framework (https://www.dotnetcurry.com/tutorials/entityframework)

.NET Framework (https://www.dotnetcurry.com/tutorials/dotnetframework)

.NET Standard & .NET Core (https://www.dotnetcurry.com/tutorials/dotnet-standard-core)

WPF (https://www.dotnetcurry.com/tutorials/wpf)

WinForms (https://www.dotnetcurry.com/tutorials/winforms)

CLOUD AND MOBILE

Microsoft Azure (https://www.dotnetcurry.com/tutorials/windows-azure)

DevOps (https://www.dotnetcurry.com/tutorials/devops)

Xamarin (https://www.dotnetcurry.com/tutorials/xamarin)

Powershell (https://www.dotnetcurry.com/tutorials/powershell)

Machine Learning & AI (https://www.dotnetcurry.com/tutorials/machine-learning-ai)

UWP & Windows Store (https://www.dotnetcurry.com/tutorials/windows-store)

Windows Phone (https://www.dotnetcurry.com/tutorials/windowsphone)

SKILL UP

Design Patterns (https://www.dotnetcurry.com/tutorials/patterns-practices)

Software Gardening (https://www.dotnetcurry.com/tutorials/software-gardening)

.NET Interview Q&A (https://www.dotnetcurry.com/tutorials/dotnetinterview)

Magazines (https://www.dotnetcurry.com/magazine/)

Books (http://www.jquerycookbook.com/)

Product Reviews (https://www.dotnetcurry.com/tutorials/product-articles-review)

FOLLOW US

- Facebook (https://www.facebook.com/dotnetcurry)
- Twitter (https://www.twitter.com/dotnetcurry)
- Github (https://github.com/dotnetcurry)
