GENERAL

# asp net core 2 0 angular 4 build from scratch a web

Published: September 19, 2026 | Index ID: #-

## Building a Web Application from Scratch with ASP.NET Core 2.0 and Angular 4

In today's fast moving web landscape, developers often look for robust, high performance stacks that can deliver modern user experiences while remaining maintainable and scalable. One combination that continues to deliver strong results is **ASP.NET Core 2.0** on the server side paired with **Angular 4**on the client. This article walks you through the entire journey of building a web application from scratch using this stack, covering everything from prerequisites to deployment, while keeping SEO best practices and human readability in mind.

## Why ASP.NET Core 2.0 and Angular 4?

ASP.NET Core 2.0 introduced a lightweight, modular framework that runs on multiple platforms. It offers fast startup, built in dependency injection, and a unified programming model for web APIs, MVC, and Razor Pages. Angular 4, part of the Angular 2+ family, brings a component based architecture, powerful data binding, and a mature ecosystem of tools.

When combined, these technologies provide:

- Separation of concerns – the backend focuses on business logic and data persistence, while the frontend handles UI and user interactions.
- Strong typing – C# on the server and TypeScript on the client both enforce type safety, reducing runtime errors.
- Rich tooling – Visual Studio, VS Code, and the Angular CLI streamline development.
- Scalability – both frameworks are designed to handle high traffic and complex workloads.

## Prerequisites and Setup

### Hardware, OS, and IDE

To get started, you'll need a development machine running Windows, macOS, or Linux. Visual Studio 2017 (or newer) with the ASP.NET and web development workload, or VS Code with the C# and Angular extensions, are highly recommended.

### Installing .NET Core SDK 2.0

Download the .NET Core 2.0 SDK appropriate for your operating system. After installation, verify it by running:

dotnet --version\n\n\nIt should output a version starting with 2.0.

### Node.js and Angular CLI

Angular 4 requires Node.js LTS 8.x or 10.x. Install Node from the official site, then add the Angular CLI globally:

npm install -g @angular/cli@1.4\n\n\nCheck the installation:

ng --version\n\n\nProject Folder

Create a clean directory for your solution, e.g., ~/Projects/AngularAspCoreDemo. Navigate into it before proceeding.

## Creating the Project from Scratch

### Step 1 – Scaffold the Backend

Open a terminal and run:

dotnet new webapi -o Backend\n\n\nThis command creates a minimal Web API project named Backend. It includes a sample controller, Swagger support, and default routing.

### Step 2 – Scaffold the Frontend

In the same solution folder, generate an Angular 4 project:

ng new Frontend --routing --style=scss\n\n\nChoose Yes when prompted to add Angular routing and SCSS for styling. The CLI creates a complete Angular workspace.

### Step 3 – Integrate the Projects

Move the Angular project into the ASP.NET Core solution by creating a ClientApp folder inside the Backend directory and copying the Frontend contents there:

mkdir Backend/ClientApp\nmv Frontend/* Backend/ClientApp/\n\n\nNow the folder structure looks like this:

Backend/\n% % Controllers/\n% % ClientApp/\n% % % src/\n% % % angular.json\n% % % package.json\n% % Startup the ASP.NET Core Startup

Edit Startup.cs to serve static files from the Angular build and to set up the default route:

public void Configure(IApplicationBuilder app, IHostingEnvironment env)\n{\n if (env.IsDevelopment())\n {\n app.UseDeveloperExceptionPage();\n }\n\n // Enable CORS for local development\n app.UseCors(builder =>\n builder.AllowAnyOrigin()\n .AllowAnyMethod()\n .AllowAnyHeader());\n\n app.UseStaticFiles();\n app.UseMvc();\n\n // Fallback to index.html for Angular routes\n app.UseSpa(spa => \n {\n spa.Options.SourcePath = "ClientApp";\n if (env.IsDevelopment())\n {\n spa.UseAngularCliServer(npmScript: "start");\n }\n });\n}\n\n\nMake sure to add the CORS policy to ConfigureServices:

services.AddCors();\nservices.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_0);
\n\n\nConfiguring the Backend API

### Adding Controllers

Create a ProductsController to expose CRUD operations:

using Microsoft.AspNetCore.Mvc;\nusing System.Collections.Generic;\n\nnamespace Backend.Controllers\n{\n [Route("api/[controller]")]\n [ApiController]\n public class ProductsController : ControllerBase\n {\n private static readonly List<Product> _products = new List<Product>()\n {\n new Product{ Id = 1, Name = "Laptop", Price = 1200 },\n new Product{ Id = 2, Name = "Smartphone", Price = 800 }\n }; \n\n [HttpGet]\n public IEnumerable<Product> Get() => _products;\n\n [HttpGet("{id}")]\n public ActionResult<Product> Get(int id)\n {\n var product = _products.Find(p => p.Id == id); \n if (product == null) return NotFound();\n return product;\n }\n\n [HttpPost]\n public ActionResult<Product> Post([FromBody] Product product)\n {\n product.Id = _products.Count + 1; \n _products.Add(product);\n return CreatedAtAction(nameof(Get), new { id = product.Id }, product); \n }\n }\n\n public class Product\n {\n public int Id { get; set; }\n public string Name { get; set; }\n public decimal Price { get; set; }\n }\n}\n\n\nDependency Injection & Entity Framework Core

For a production app, replace the in  memory list with a database. Add Microsoft.EntityFrameworkCore.SqlServer via NuGet, then define a ProductContext and register it:

public class ProductContext : DbContext\n{\n public ProductContext(DbContextOptions<ProductContext> options) : base(options) { }\n\n  public DbSet<Product> Products { get; set; }\n}\n\n\nIn ConfigureServices:

services.AddDbContext<ProductContext>(options =>\n options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));\n\n\nUpdate the controller to use ProductContext via constructor injection.

### Cross  Origin Resource Sharing (CORS)

Since the Angular app runs on a different port during development, enable CORS in ConfigureServices:

services.AddCors(options =>\n{\n options.AddPolicy("AllowAll",\n builder =>\n {\n builder.AllowAnyOrigin()\n .AllowAnyMethod()\n .AllowAnyHeader();\n });\n});\n\n\nAnd apply it in Configure:

app.UseCors("AllowAll");\n\n\nIntegrating Angular 4 Frontend

### Project Structure Overview

Angular 4 organizes code into modules, components, services, and routing. The src/app folder contains the main app.module.ts, which imports other modules and declares components.

### Creating a Product Service

Generate a service to communicate with the API:

ng generate service services/product\n\n\nImplement HTTP methods using HttpClient:

import { Injectable } from '@angular/core';\nimport { HttpClient } from '@angular/common/http';\nimport { Observable } from 'rxjs';\nimport { Product } from '../models/product';\n\n@Injectable({\n providedIn: 'root'\n})\nexport class ProductService {\n private apiUrl = '/api/products';\n\n constructor(private http: HttpClient) { }\n\n getProducts(): Observable<Product[]> {\n return this.http.get<Product[]>(this.apiUrl);\n }\n\n getProduct(id: number): Observable<Product> {\n return this.http.get<Product>(`${this.apiUrl}/${id}`);\n }\n\n addProduct(product: Product): Observable<Product> {\n return this.http.post<Product>(this.apiUrl, product);\n }\n}\n\n\nBuilding Components

Create a ProductListComponent to display all products:

ng generate component components/product-list\n\n\nIn product-list.component.ts, inject ProductService and fetch data in ngOnInit:

import { Component, OnInit } from '@angular/core';\nimport { ProductService } from '../../services/product.service';\nimport { Product } from '../../models/product';\n\n@Component({\n selector: 'app-product-list',\n templateUrl: './product-list.component.html',\n styleUrls: ['./product-list.component.scss']\n})\nexport class ProductListComponent implements OnInit {\n products: Product[] = [];\n\n constructor(private productService: ProductService) {}\n\n ngOnInit(): void {\n this.productService.getProducts().subscribe(data => this.products = data);\n }\n}\n\n\nRouting Setup

Define routes in app-routing.module.ts:

## Baca Juga (Artikel Terkait)

- **advanced engineering electromagnetics solution manual edition hardcover by balanis constantine a published by wiley**

  URL: http://dtrainderailed.com/advanced-engineering-electromagnetics-solution-manual-edition-hardcover-by-balanis-constantine-a-published-by-wiley.pdf

- **advanced engineering electromagnetics solutions manual**

  URL: http://dtrainderailed.com/advanced-engineering-electromagnetics-solutions-manual.pdf

- **advanced engineering electromagnetics wiley 1989 grading**

  URL: http://dtrainderailed.com/advanced-engineering-electromagnetics-wiley-1989-grading.pdf

- **advanced engineering fluid mechanics by biswas**

  URL: http://dtrainderailed.com/advanced-engineering-fluid-mechanics-by-biswas.pdf