Building Web App using ASP.NET Web API Angular 7 and SQL Server

by Didin J. on Jan 30, 2019

> The comprehensive step by step tutorial on building Web Application using ASP.NET Web API, Angular 7 and Microsoft SQL Server

The comprehensive step by step tutorial on building Web Application using ASP.NET Web API, Angular 7 and Microsoft SQL Server. In this tutorial, we will create a RESTful API web service using ASP.NET Core Web API then create a front-end application with Angular 7. For the backend, we use Microsoft SQL Server 2017 Express Edition with the additional database Northwind. This is our first ASP.NET tutorial, so there will be many shortcomings and a lot of discussions.

Table of Contents:

- Create Helpers Class for Mapping DTO with Model Classes
- Create Repositories for CRUD (Create, Read, Update Delete) Operations
- Create Controller for CRUD Operations
- Test API using Postman
- Install or Update Angular 7 CLI and Create Application
- Add Routes for Navigation between Angular Pages/Component
- Create Service for Accessing RESTful API
- Display List of Suppliers using Angular 7 Material
- Show and Delete Supplier Details using Angular 7 Material
- Add a Supplier using Angular 7 Material
- Edit a Supplier using Angular 7 Material
- Run and Test the ASP.NET Core Web API and Angular 7 CRUD Web Application

---

The following tools, frameworks, package, and modules are required for this tutorial:

- ASP.NET Core SDK (\)
- Visual Studio Code (\)
- Node.js (\)
- Angular 7 (\)
- SQL Server 2017 Express
- Postman
- Command prompt (CMD)

We assume that you already install all above required tools. Now, we have to check that DotNet SDK already installed and added to the path. Open command prompt then type this command to make sure DotNet installed and runnable via command prompt.

```
dotnet --version
```

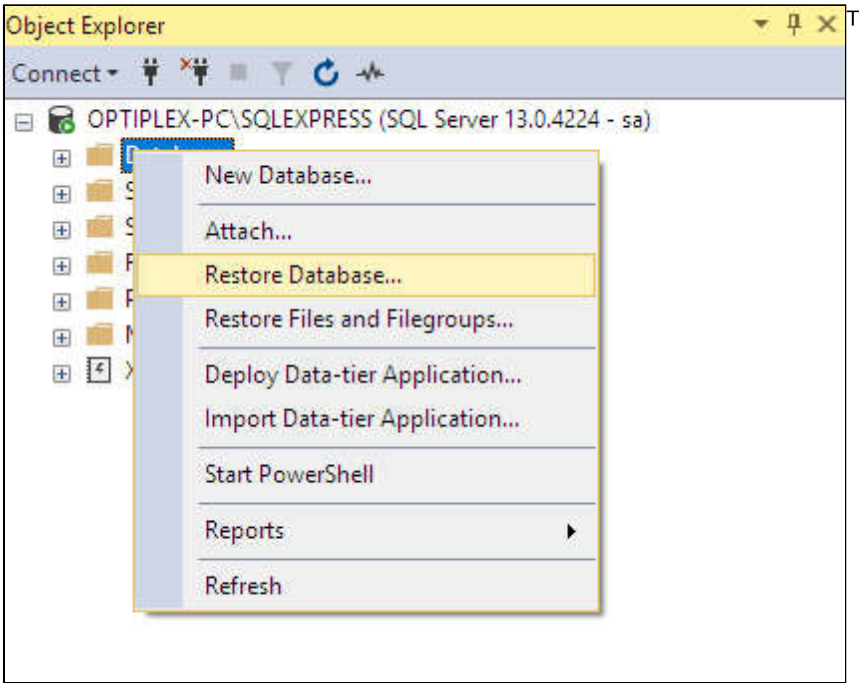
You will get this result.

```
2.1.403
```

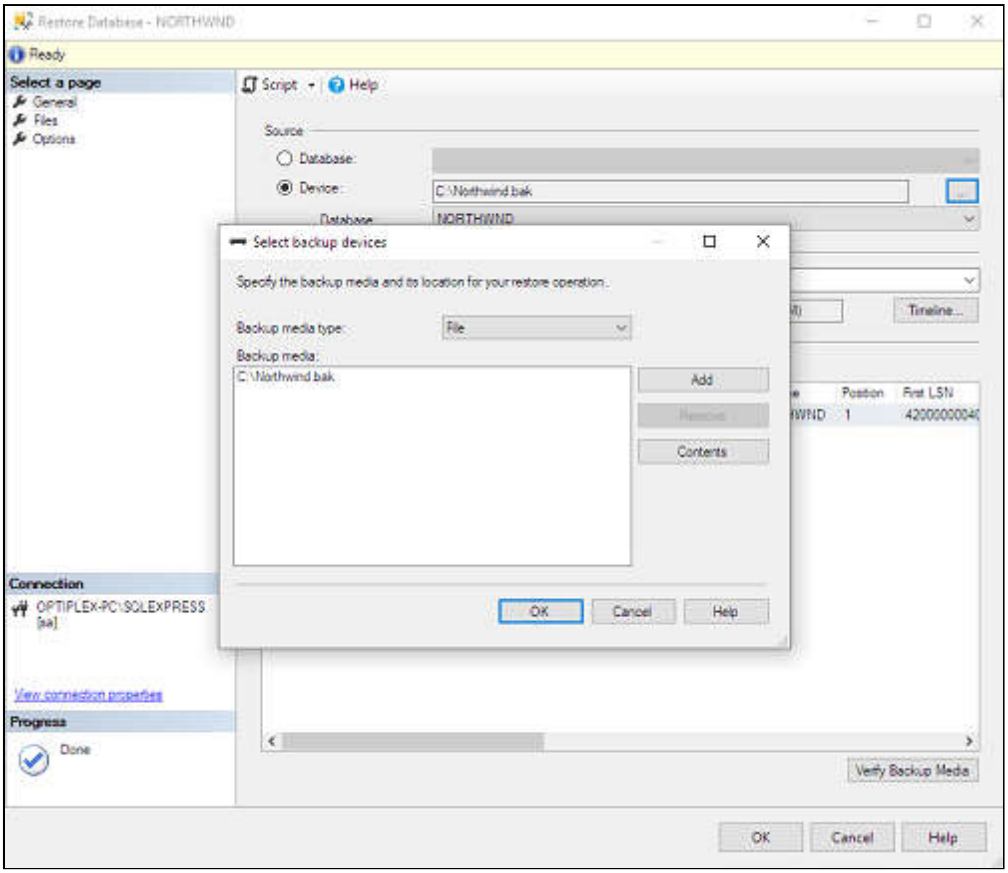
We are using the DotNet Framework SDK version 2.1.403. That's mean you're ready to move to the main steps.

1. Download and Install Northwind Sample Database

To download the Northwind sample database for SQL Server 2017, open your browser then go to this link (https://northwinddatabase.codeplex.com/). After download, extract to your root folder (for example `C:\\`). Next, open Microsoft SQL Server Management Studio. Login to your local Server then right-click Database in the Object Explorer Window then choose Restore Database.



Choose Device then Click `...` button. The select device dialog will open then click Add button, browse to the `.bak` file that previously extracted then click `OK` button then click again `OK` button.



Click the `OK` button to Restore the Northwind Database. Now, you should see the Northwind Database in the Database.

2. Create and Configure a new ASP.NET Web API Application

A slightly different of ASP.NET Core application creation, we will use the command prompt. Of course, you can use the more easiest way when using Microsoft Visual Studio. In the command prompt go to your projects folder then type this command.

```
dotnet  new webapi -o AspNetAngular -n AspNetAngular
```

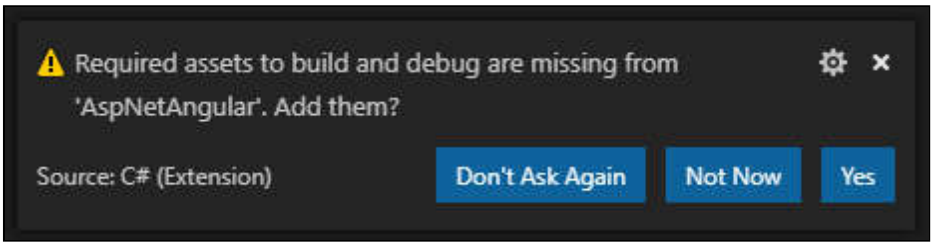
You will the output like below in the Command Prompt.

```
Processing post-creation actions...
Running 'dotnet restore' on AspNetAngular\AspNetAngular.csproj...
  Restoring packages for C:\Users\DIDIN\Projects\AspNetAngular\AspNetAngular.csproj...
  Generating MSBuild file C:\Users\DIDIN\Projects\AspNetAngular\obj\AspNetAngular.csproj.nuget.g.props.
  Generating MSBuild file C:\Users\DIDIN\Projects\AspNetAngular\obj\AspNetAngular.csproj.nuget.g.targets.
  Restore completed in 5.82 sec for C:\Users\DIDIN\Projects\AspNetAngular\AspNetAngular.csproj.

Restore succeeded.
```

That's mean, you can open directly as a folder from Visual Studio Code as well as CSharp Project/Solution from Microsoft Visual Studio. Or you can call this new ASP.NET Core Web API Project by type this command.

```
cd AspNetAngular
code .
```

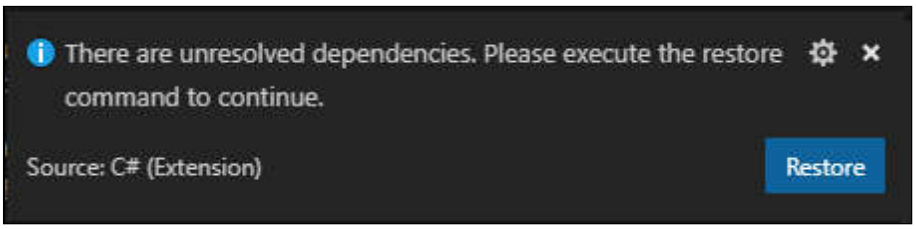
The project will be opened by Visual Studio Code. If you get a notification like below just click `Yes` button.



Next, open terminal from the menu or press Ctrl+Shift+`. Run these commands to installs all required packages.

```
dotnet add package Microsoft.EntityFrameworkCore.Tools -v 2.1.2
dotnet add package Microsoft.EntityFrameworkCore.SqlServer -v 2.1.2
dotnet add package Microsoft.EntityFrameworkCore.SqlServer.Design -v 1.1.6
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design -v 2.1.5
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Tools -v 2.0.4
dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection -v 5.0.1
```

If you see a notification like below, just click Restore button.



Now, you will have these packages with the right version installed shown in `AspNetAngular.csproj` file.

```
<ItemGroup>
  <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="6.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.App" />
  <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.1.2" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer.Design" Version="1.1.6" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.1.2">
    <IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>
    <PrivateAssets>all</PrivateAssets>
  </PackageReference>
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.5" />
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.4" />
</ItemGroup>
```

Next, build the DotNet application to make sure there's no error in package dependencies.

```
dotnet build
```

You must see this output when everything on the right path.

```
Build succeeded.
    0 Warning(s)
    0 Error(s)
```

Next, we have to configure connections to Microsoft SQL Server Database. First, open and edit `appsettings.json` then add these lines before `logging` JSON object.

```
"ConnectionStrings": {
  "SQLConnection": "Server=.;Database=NORTHWND;Trusted_Connection=True;User Id=sa;Password=q;Integrated Sec
urity=false;MultipleActiveResultSets=true"
},
```

Open and edit `Startup.cs` then add this line inside `ConfigureServices` bracket.

```
services.AddDbContext<NORTHWNDContext>(options => options.UseSqlServer(Configuration.GetConnectionString("S
QLConnection")));
```

Next, add this line to enable CORS after above line.

```
services.AddCors();
```

Also, add this line inside `Configure` bracket.

```
app.UseCors(x => x.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());
```

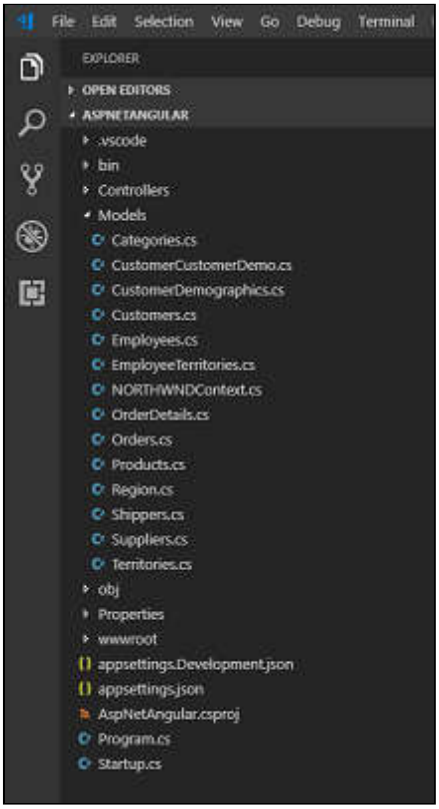
Don't forget to import the library or module that added in the Configuration.

3. Generate Models from Microsoft SQL Server Database

We will use the Code Generation Tools package to generate all models that represent all tables in the Northwind Database. Run this command to generate the models and their context.

```
dotnet ef dbcontext scaffold "Server=.;Database=NORTHWND;Trusted_Connection=True;User Id=sa;Password=q;Integrated Security=false;" Microsoft.EntityFrameworkCore.SqlServer -o Models
```

You will see the generated Models and Context inside the Models folder.



If there's a lot of warning or error that comes from IDE linter, just re-open the Visual Studio Code. As you see in the table of contents, we will use only the Supplier model for this tutorial. The generated suppliers look like this.

```
using System;
using System.Collections.Generic;

namespace AspNetAngular.Models
{
    public partial class Suppliers
    {
        public Suppliers()
        {
            Products = new HashSet<Products>();
        }

        public int SupplierId { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }
        public string HomePage { get; set; }

        public ICollection<Products> Products { get; set; }
    }
}
```

That model declared in the `NORTHWNDContext.cs` the file inside Models folder.

```
public virtual DbSet<Suppliers> Suppliers { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Suppliers>(entity =>
  {
      entity.HasKey(e => e.SupplierId);

      entity.HasIndex(e => e.CompanyName)
          .HasName("CompanyName");

      entity.HasIndex(e => e.PostalCode)
          .HasName("PostalCode");

      entity.Property(e => e.SupplierId).HasColumnName("SupplierID");

      entity.Property(e => e.Address).HasMaxLength(60);

      entity.Property(e => e.City).HasMaxLength(15);

      entity.Property(e => e.CompanyName)
          .IsRequired()
          .HasMaxLength(40);

      entity.Property(e => e.ContactName).HasMaxLength(30);

      entity.Property(e => e.ContactTitle).HasMaxLength(30);

      entity.Property(e => e.Country).HasMaxLength(15);

      entity.Property(e => e.Fax).HasMaxLength(24);

      entity.Property(e => e.HomePage).HasColumnType("ntext");

      entity.Property(e => e.Phone).HasMaxLength(24);

      entity.Property(e => e.PostalCode).HasMaxLength(10);

      entity.Property(e => e.Region).HasMaxLength(15);
  });
}
```

4. Create DTO (Data Transfer Object) for Request Body and Response

To specify the request body and response fields, we will use a Data Transfer Object (DTO). For that, create a new DTOs folder and `AddSupplierDto.cs, EditSupplierDto.cs, SupplierResponse.cs` files inside that folder. Next, open and edit `AddSupplierDto.cs` then Replace all codes with this.

```
using System;
using System.ComponentModel.DataAnnotations;

namespace AspNetAngular.Dtos
{
    public class AddSupplierDto
    {
        [Required]
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }
        public string HomePage { get; set; }
    }
}
```

Next, open and edit `EditSupplierDto.cs` then Replace all codes with this.

```
using System;
using System.ComponentModel.DataAnnotations;

namespace AspNetAngular.Dtos
{
    public class EditSupplierDto
    {
        [Required]
        public int SupplierId { get; set; }
        [Required]
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }
        public string HomePage { get; set; }
    }
}
```

Next, open and edit `SupplierResponseDto.cs` then Replace all codes with this.

```
namespace AspNetAngular.Dtos
{
    public class SupplierResponseDto
    {
        public int SupplierId { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string PostalCode { get; set; }

    }
}
```

5. Create Helpers Class for Mapping DTO with Model Classes

To create a helper for mapping DTO to Model classes, create the folder `Helpers` in the root of the project folder then create a file `AutoMapperProfile.cs` inside that new folder. Open and edit this new file then replace all codes with this.

```
using AspNetAngular.Dtos;
using AspNetAngular.Models;
using AutoMapper;

namespace AspNetAngular.Helpers
{
    public class AutoMapperProfiles: Profile
    {
        public AutoMapperProfiles()
        {
            CreateMap<AddSupplierDto, Suppliers>();
            CreateMap<EditSupplierDto, Suppliers>();
            CreateMap<Suppliers, SupplierResponseDto>();
        }
    }
}
```

Next, open and edit again `Startup.cs` then add this line inside `ConfigureServices`.

```
services.AddAutoMapper();
```

Don't forget to auto import the library in `ConfigureServices`.

6. Create Repositories for CRUD (Create, Read, Update Delete) Operations

To create a repository class and interface, we have to create a folder `Repositories` in the root of the project folder. Next, create an interface file inside that folder with the name `IDataRepository.cs` then replace all codes with this.

```
using System.Threading.Tasks;

namespace AspNetAngular.Repositories
{
    public interface IDataRepository<T> where T : class
    {
        void Add(T entity);
        void Update(T entity);
        void Delete(T entity);
        Task<T> SaveAsync(T entity);
    }
}
```

Next, create a file inside that folder with the name `DataRepository.cs` then replace all codes with this.

```
using System.Threading.Tasks;
using AspNetAngular.Models;

namespace AspNetAngular.Repositories
{
    public class DataRepository<T> : IDataRepository<T> where T : class
    {
        private readonly NORTHWNDContext _context;

        public DataRepository(NORTHWNDContext context)
        {
            _context = context;
        }

        public void Add(T entity)
        {
            _context.Set<T>().Add(entity);
        }

        public void Update(T entity)
        {
            _context.Set<T>().Update(entity);
        }

        public void Delete(T entity)
        {
            _context.Set<T>().Remove(entity);
        }

        public async Task<T> SaveAsync(T entity)
        {
            await _context.SaveChangesAsync();
            return entity;
        }
    }
}
```

Next, open and edit again `Startup.cs` then add this line inside `ConfigureServices` bracket.

```
services.AddScoped(typeof(IDataRepository < > ), typeof(DataRepository < > ));
```

Also, import the required classes and libraries.

7. Create Controller for CRUD Operations

Now, we will show you how the API available via Controller. We will generate a controller for Supplier model. Before generate, we have to install the tools for it. Run this command in the terminal of Visual Studio Code.

```
dotnet tool install --global dotnet-aspnet-codegenerator --version 2.2.1
```

Just run this command to generate it.

```
dotnet aspnet-codegenerator controller -name SupplierController -api -async -m AspNetAngular.Models.Supplie
rs -dc NORTHWNDContext -namespace AspNetAngular.Controllers -outDir Controllers
```

Now, we have a Supplier's controller that looks like this.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using AspNetAngular.Models;

namespace AspNetAngular.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class SupplierController : ControllerBase
    {
        private readonly NORTHWNDContext _context;

        public SupplierController(NORTHWNDContext context)
        {
            _context = context;
        }

        // GET: api/Supplier
        [HttpGet]
        public IEnumerable<Suppliers> GetSuppliers()
        {
            return _context.Suppliers;
        }

        // GET: api/Supplier/5
        [HttpGet("{id}")]
        public async Task<IActionResult> GetSuppliers([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var suppliers = await _context.Suppliers.FindAsync(id);

            if (suppliers == null)
            {
                return NotFound();
            }

            return Ok(suppliers);
        }

        // PUT: api/Supplier/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutSuppliers([FromRoute] int id, [FromBody] Suppliers suppliers)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
```

```
        }

        if (id != suppliers.SupplierId)
        {
            return BadRequest();
        }

        _context.Entry(suppliers).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!SuppliersExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/Supplier
    [HttpPost]
    public async Task<IActionResult> PostSuppliers([FromBody] Suppliers suppliers)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        _context.Suppliers.Add(suppliers);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetSuppliers", new { id = suppliers.SupplierId }, suppliers);
    }

    // DELETE: api/Supplier/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteSuppliers([FromRoute] int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var suppliers = await _context.Suppliers.FindAsync(id);
        if (suppliers == null)
        {
            return NotFound();
```

```
            }

            _context.Suppliers.Remove(suppliers);
            await _context.SaveChangesAsync();

            return Ok(suppliers);
        }

        private bool SuppliersExists(int id)
        {
            return _context.Suppliers.Any(e => e.SupplierId == id);
        }
    }
}
```

As you see that generated Controllers using original fields from the Suppliers Model. Next, we will use our previous created DTO for custom Object save, edit and their response. Open and edit `Controllers/SupplierController.cs` then add this variable after context variable.

```
private readonly IMapper _mapper;
private readonly IDataRepository<Suppliers> _repo;
```

Also, inject to the SupplierController constructor.

```
public SupplierController(NORTHWNDContext context, IMapper mapper, IDataRepository<Suppliers> repo)
{
    _context = context;
    _mapper = mapper;
    _repo = repo;
}
```

Don't forget to import all required libraries or classes. Next, replace the HTTPPost method with these codes.

```
// POST: api/Supplier
[HttpPost]
public async Task<IActionResult> PostSuppliers([FromBody] AddSupplierDto addSupplierDto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var preSupplier = _mapper.Map<Suppliers>(addSupplierDto);
    _repo.Add(preSupplier);
    var saveSupplier = await _repo.SaveAsync(preSupplier);
    var supplierResponse = _mapper.Map<SupplierResponseDto>(saveSupplier);

    return StatusCode(201, new { supplierResponse });
}
```

Next, replace the HTTPPut method with these codes.

```
// PUT: api/Supplier/5
[HttpPut("{id}")]
public async Task<IActionResult> PutSuppliers([FromRoute] int id, [FromBody] EditSupplierDto editSupplierDt
o)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != editSupplierDto.SupplierId)
    {
        return BadRequest();
    }

    var preSupplier = _mapper.Map<Suppliers>(editSupplierDto);
    _repo.Update(preSupplier);
    await _repo.SaveAsync(preSupplier);

    return NoContent();
}
```

Finally, replace the `HTTPDelete` method with this because we will delete manually busing SQL Query the related table.

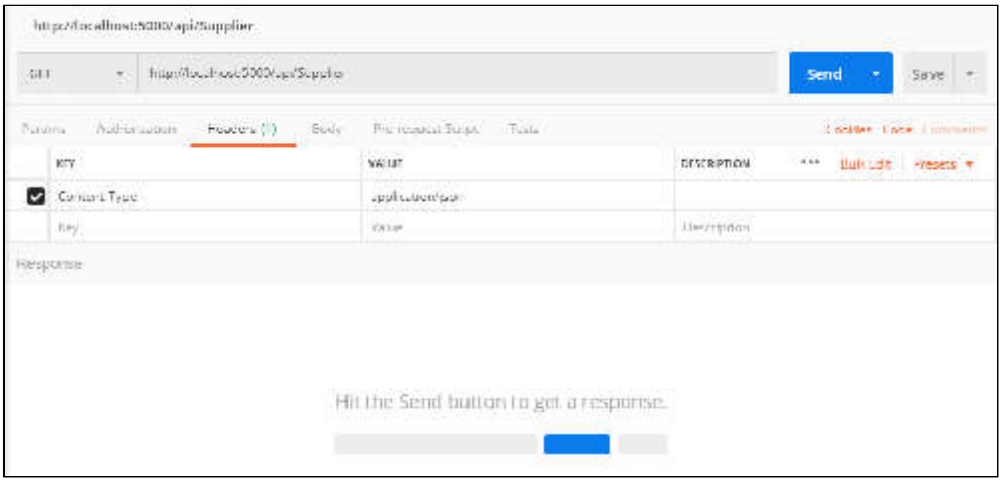
```
// DELETE: api/Supplier/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteSuppliers([FromRoute] int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var suppliers = await _context.Suppliers.FindAsync(id);

    if (suppliers == null)
    {
        return NotFound();
    }

    _context.Database.ExecuteSqlCommand("DELETE FROM [Order Details] WHERE ProductID IN (SELECT ProductID F
ROM Products WHERE SupplierID = @supplierId)",
        new SqlParameter("@supplierId", suppliers.SupplierId));

    _context.Database.ExecuteSqlCommand("DELETE FROM Products WHERE SupplierID = @supplierId",
        new SqlParameter("@supplierId", suppliers.SupplierId));

    _context.Suppliers.Remove(suppliers);
    await _context.SaveChangesAsync();

    return Ok(suppliers);
}
```

8. Test API using Postman

Now, we have to run the ASP.NET Core Web API from the Terminal by typing this command.

```
dotnet watch run
```

Watch keyword is the additional command for monitoring any change in the codes then reloading the ASP.NET Core Web API application. Next, open or run the Postman application. Use the GET method and fill the right column after the GET method with `localhost:5000/api/Supplier`, Headers key with `Content-Type`, Headers value with `application/json`.



Now, click the Send button and you should see this result for successful GET Supplier.

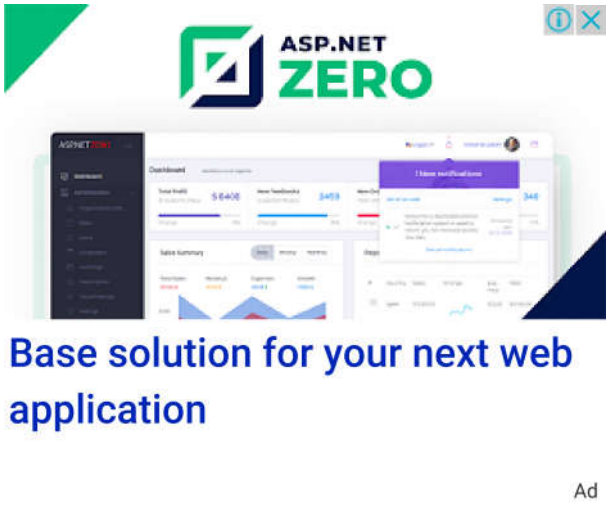
```
[
    {
        "supplierId": 1,
        "companyName": "Exotic Liquids",
        "contactName": "Charlotte Cooper",
        "contactTitle": "Purchasing Manager",
        "address": "49 Gilbert St.",
        "city": "London",
        "region": null,
        "postalCode": "EC1 4SD",
        "country": "UK",
        "phone": "(171) 555-2222",
        "fax": null,
        "homePage": null,
        "products": []
    },
    {
        "supplierId": 2,
        "companyName": "New Orleans Cajun Delights",
        "contactName": "Shelley Burke",
        "contactTitle": "Order Administrator",
        "address": "P.O. Box 78934",
        "city": "New Orleans",
        "region": "LA",
        "postalCode": "70117",
        "country": "USA",
        "phone": "(100) 555-4822",
        "fax": null,
        "homePage": "#CAJUN.HTM#",
        "products": []
    },
    ...
]
```


To get single supplier by supplier ID, just change the URL to this `localhost:5000/api/Supplier/2`. If found, you will see the result of a single supplier. Next, for POST a Supplier changes the method to `POST`, leave URL same as the GET supplier list then fill the body with a raw JSON object.

```
{
  "CompanyName": "Djamware Inc.",
  "ContactName": "Didin J.",
  "ContactTitle": "CEO",
  "Address": "Whereever Road 123",
  "City": "Bandung",
  "Region": "JBR",
  "PostalCode": "12345",
  "Country": "Indonesia",
  "Phone": "(022) 123-4567890",
  "Fax": "(022) 123-4567890",
  "HomePage": "https://www.djamware.com"
}
```

You will see this result with status 201.

```
{
    "supplierResponse": {
        "companyName": "Djamware Inc.",
        "contactName": "Didin J.",
        "contactTitle": "CEO",
        "address": "Whereever Road 123",
        "city": "Bandung",
        "region": "JBR",
        "postalCode": "12345",
        "country": "Indonesia",
        "phone": "(022) 123-4567890",
        "fax": "(022) 123-4567890",
        "homePage": "https://www.djamware.com"
    }
}
```



To update the Supplier data, change the method to `PUT` and URL to `localhost:5000/api/Supplier/30` then add a supplier id field to the raw body.

```
{
  "SupplierId": 30,
  "CompanyName": "Djamware.com",
  "ContactName": "Didin J.",
  "ContactTitle": "Engineer",
  "Address": "Whereever Road 123",
  "City": "Garut",
  "Region": "JBR",
  "PostalCode": "12345",
  "Country": "Indonesia",
  "Phone": "(022) 123-4567890",
  "Fax": "(022) 123-4567890",
  "HomePage": "https://www.djamware.com"
}
```

The response should be 204 (No Content) for a successful request. Next, for delete a supplier, simply change the method to `DELETE`, keep the URL as the previous method and change the body to none. You should see the 200 response and data that deleted in the response body.

9. Install or Update Angular 7 CLI and Create Application

Before installing the Angular 7 CLI, make sure you have installed Node.js https://nodejs.org and can open Node.js command prompt. Next, open the Node.js command prompt then type this command to install Angular 7 CLI.

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

Next, create an Angular 7 application by typing this command in the root of ASP.NET Core application/project directory.

```
ng new client
```

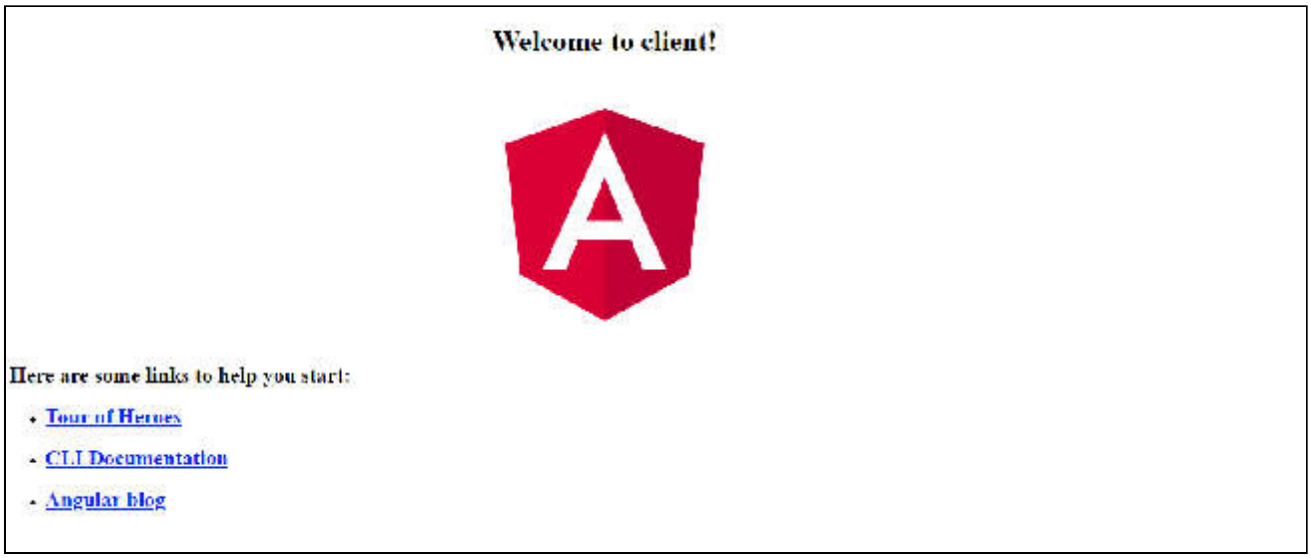
Where `client` is the name of the Angular 7 application. You can specify your own name, we like to name it `client` because it's put inside ASP.NET Core Project directory. If there's a question, we fill them with `Y` and `SASS`. Next, go to the newly created Angular 7 application.

```
cd client
```

Run the Angular 7 application for the first time.

```
ng serve
```

Now, go to `localhost:4200` and you should see this page.



To stop the Angular 7 application, just press `CTRL+C` keys.

10. Add Routes for Navigation between Angular 7 Pages/Component

On the previous steps, we have to added Angular 7 Routes when answering the questions. Now, we just added the required pages for CRUD (Create, Read, Update, Delete) Supplier data. Type this commands to add the Angular 7 components or pages.

```
ng g component supplier
ng g component supplier-detail
ng g component supplier-add
ng g component supplier-edit
```

Open `src/app/app.module.ts` then you will see those components imported and declared in `@NgModule` declarations. Next, open and edit `src/app/app-routing.module.ts` then add this imports.

```
import { SupplierComponent } from './supplier/supplier.component';
import { SupplierDetailComponent } from './supplier-detail/supplier-detail.component';
import { SupplierAddComponent } from './supplier-add/supplier-add.component';
import { SupplierEditComponent } from './supplier-edit/supplier-edit.component';
```

Add these arrays to the existing routes constant.

```
const routes: Routes = [
  {
    path: 'supplier',
    component: SupplierComponent,
    data: { title: 'List of Suppliers' }
  },
  {
    path: 'supplier-details/:id',
    component: SupplierDetailComponent,
    data: { title: 'Supplier Details' }
  },
  {
    path: 'supplier-add',
    component: SupplierAddComponent,
    data: { title: 'Add Supplier' }
  },
  {
    path: 'supplier-edit/:id',
    component: SupplierEditComponent,
    data: { title: 'Edit Supplier' }
  },
  { path: '',
    redirectTo: '/supplier',
    pathMatch: 'full'
  }
];
```

Open and edit `src/app/app.component.html` and you will see existing router outlet. Next, modify this HTML page to fit the CRUD page.

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="
cvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4yb
DE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMw
djIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xeiIgLz4KICAgIDxwYXRoIGZpbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xxem0xNyA4My4zaC0zNGwxNy00MC45IDE
E3IDQwLjl6IiAvPgogIDwvc3ZnPg==">
</div>

<div class="container">
  <router-outlet></router-outlet>
</div>
```

Open and edit `src/app/app.component.scss` then replace all SCSS codes with this.

```
.container {
  padding: 20px;
}
```

11. Create Service for Accessing RESTful API

To access ASP.NET Core Web API from Angular 7 application, we have to create an Angular 7 Service first. Type this command to create it.

```
ng g service api
```

We have to register the `HttpClient` module before using it in the Service. Open and edit `src/app/app.module.ts` then add this import.

```
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
```

Add them to the `@NgModule` imports array.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
  AppRoutingModule
],
```

To specify the type of response object from the ASP.NET Core Web API, we have to create a class for it. Create a new file `src/app/supplier.ts` then add this codes.

```
export class Supplier {
    supplierId: number;
    companyName: string;
    contactName: string;
    contactTitle: string;
    address: string;
    city: string;
    region: string;
    postalCode: string;
    country: string;
    phone: string;
    fax: string;
    homePage: string;
}
```

Next, open and edit `src/app/api.service.ts` then add this imports.

```
import { Observable, of, throwError } from 'rxjs';
import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';
import { catchError, tap, map } from 'rxjs/operators';
import { Supplier } from './supplier';
```

Add these constants before the `@Injectable`.

```
const httpOptions = {
  headers: new HttpHeaders({'Content-Type': 'application/json'})
};
const apiUrl = 'http://localhost:5000/api/';
```

Inject `HttpClient` module to the constructor.

```
constructor(private http: HttpClient) { }
```

Add the error handler function.

```
private handleError<T> (operation = 'operation', result?: T) {
  return (error: any): Observable<T> => {

    // TODO: send the error to remote logging infrastructure
    console.error(error); // log to console instead

    // Let the app keep running by returning an empty result.
    return of(result as T);
  };
}
```

Add all CRUD (create, read, update, delete) functions of suppliers data.

```
getSuppliers (): Observable<Supplier[]> {
  return this.http.get<Supplier[]>(apiUrl)
    .pipe(
      tap(heroes => console.log('fetched Suppliers')),
      catchError(this.handleError('getSuppliers', []))
    );
}

getSupplier(id: number): Observable<Supplier> {
  const url = `${apiUrl}/${id}`;
  return this.http.get<Supplier>(url).pipe(
    tap(_ => console.log(`fetched Supplier id=${id}`)),
    catchError(this.handleError<Supplier>(`getSupplier id=${id}`))
  );
}

addSupplier (supplier: any): Observable<Supplier> {
  return this.http.post<Supplier>(apiUrl, supplier, httpOptions).pipe(
    tap((supplierRes: Supplier) => console.log(`added Supplier w/ id=${supplierRes.supplierId}`)),
    catchError(this.handleError<Supplier>('addSupplier'))
  );
}

updateSupplier (id: number, supplier: any): Observable<any> {
  const url = `${apiUrl}/${id}`;
  return this.http.put(url, supplier, httpOptions).pipe(
    tap(_ => console.log(`updated Supplier id=${id}`)),
    catchError(this.handleError<any>('updateSupplier'))
  );
}

deleteSupplier (id: number): Observable<Supplier> {
  const url = `${apiUrl}/${id}`;
  return this.http.delete<Supplier>(url, httpOptions).pipe(
    tap(_ => console.log(`deleted Supplier id=${id}`)),
    catchError(this.handleError<Supplier>('deleteSupplier'))
  );
}
```

12. Display List of Suppliers using Angular 7 Material

To display a list of suppliers to the Angular 7 template. First, open and edit `src/app/supplier/supplier.component.ts` then add this imports.

```
import { ApiService } from '../api.service';
```

Next, inject the API Service to the constructor.

```
constructor(private api: ApiService) { }
```

Next, for user interface (UI) we will use Angular 7 Material and CDK. There's a CLI for generating a Material component like Table as a component, but we will create or add the Table component from scratch to existing component. Type this command to install Angular 7 Material.

```
ng add @angular/material
```

If there are some questions, answer them like below.

```
? Choose a prebuilt theme name, or "custom" for a custom theme: Purple/Green       [ Preview: https://mater
ial.angular.i
o?theme=purple-green ]
? Set up HammerJS for gesture recognition? Yes
? Set up browser animations for Angular Material? Yes
```

Next, we have to register all required Angular Material components or modules to `src/app/app.module.ts`. Open and edit that file then add this imports.

```
import {
  MatInputModule,
  MatPaginatorModule,
  MatProgressSpinnerModule,
  MatSortModule,
  MatTableModule,
  MatIconModule,
  MatButtonModule,
  MatCardModule,
  MatFormFieldModule } from '@angular/material';
```

Also, modify `FormsModule` import to add `ReactiveFormsModule`.

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

Register the above modules to `@NgModule` imports.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
  AppRoutingModule,
  ReactiveFormsModule,
  BrowserAnimationsModule,
  MatInputModule,
  MatTableModule,
  MatPaginatorModule,
  MatSortModule,
  MatProgressSpinnerModule,
  MatIconModule,
  MatButtonModule,
  MatCardModule,
  MatFormFieldModule
],
```

Next, back to `src/app/supplier/supplier.component.ts` then add this imports.

```
import { Supplier } from '../supplier';
```

Declare the variables of Angular Material Table Data Source before the constructor.

```
displayedColumns: string[] = ['supplierId', 'companyName', 'contactName'];
data: Supplier[] = [];
isLoadingResults = true;
```

Modify the `ngOnInit` function to get a list of suppliers immediately.

```
ngOnInit() {
  this.api.getSuppliers()
  .subscribe(res => {
    this.data = res;
    console.log(this.data);
    this.isLoadingResults = false;
  }, err => {
    console.log(err);
    this.isLoadingResults = false;
  });
}
```

Next, open and edit `src/app/supplier/supplier.component.html` then replace all HTML tags with this Angular 7 Material tags.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" [routerLink]="['/supplier-add']"><mat-icon>add</mat-icon></a>
  </div>
  <div class="mat-elevation-z8">
    <table mat-table [dataSource]="data" class="example-table"
           matSort matSortActive="CompanyName" matSortDisableClear matSortDirection="asc">

      <!-- Supplier ID Column -->
      <ng-container matColumnDef="supplierId">
        <th mat-header-cell *matHeaderCellDef>Supplier ID</th>
        <td mat-cell *matCellDef="let row">{{row.supplierId}}</td>
      </ng-container>

      <!-- Company Name Column -->
      <ng-container matColumnDef="companyName">
        <th mat-header-cell *matHeaderCellDef>Company Name</th>
        <td mat-cell *matCellDef="let row">{{row.companyName}}</td>
      </ng-container>

      <!-- Contact Name Column -->
      <ng-container matColumnDef="contactName">
        <th mat-header-cell *matHeaderCellDef>Contact Name</th>
        <td mat-cell *matCellDef="let row">{{row.contactName}}</td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
      <tr mat-row *matRowDef="let row; columns: displayedColumns;" [routerLink]="['/supplier-details/', ro
w.supplierId]"></tr>
    </table>
  </div>
</div>
```

Finally, to make a little UI adjustment, open and edit `src/app/supplier/supplier.component.css` then add this CSS codes.

```
/* Structure */
.example-container {
    position: relative;
    padding: 5px;
  }

  .example-table-container {
    position: relative;
    max-height: 400px;
    overflow: auto;
  }

  table {
    width: 100%;
  }

  .example-loading-shade {
    position: absolute;
    top: 0;
    left: 0;
    bottom: 56px;
    right: 0;
    background: rgba(0, 0, 0, 0.15);
    z-index: 1;
    display: flex;
    align-items: center;
    justify-content: center;
  }

  .example-rate-limit-reached {
    color: #980000;
    max-width: 360px;
    text-align: center;
  }

  /* Column Widths */
  .mat-column-number,
  .mat-column-state {
    max-width: 64px;
  }

  .mat-column-created {
    max-width: 124px;
  }

  .mat-flat-button {
    margin: 5px;
  }
```

13. Show and Delete Supplier Details using Angular 7 Material

To show supplier details after click or tap on the one of a row inside the Angular 7 Material table, open and edit `src/app/supplier-detail/supplier-detail.component.ts` then add this imports.

```
import { ActivatedRoute, Router } from '@angular/router';
import { ApiService } from '../api.service';
import { Supplier } from '../supplier';
```

Inject above modules to the constructor.

```
constructor(private route: ActivatedRoute, private api: ApiService, private router: Router) { }
```

Declare the variables before the constructor for hold supplier data that get from the API.

```
supplier: Supplier = {
  supplierId: null,
  companyName: '',
  contactName: '',
  contactTitle: '',
  address: '',
  city: '',
  region: '',
  postalCode: '',
  country: '',
  phone: '',
  fax: '',
  homePage: ''
};
isLoadingResults = true;
```

Add a function for getting Supplier data from the API.

```
getSupplierDetails(id) {
  this.api.getSupplier(id)
    .subscribe(data => {
      this.supplier = data;
      console.log(this.supplier);
      this.isLoadingResults = false;
    });
}
```

Call that function when the component is initiated.

```
ngOnInit() {
  this.getSupplierDetails(this.route.snapshot.params['id']);
}
```

Add this function for delete supplier.

```
deleteSupplier(id: number) {
  this.isLoadingResults = true;
  this.api.deleteSupplier(id)
    .subscribe(res => {
        this.isLoadingResults = false;
        this.router.navigate(['/supplier']);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      }
    );
}
```

For the view, open and edit `src/app/supplier-detail/supplier-detail.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" [routerLink]="['/supplier']"><mat-icon>list</mat-icon></a>
  </div>
  <mat-card class="example-card">
    <mat-card-header>
      <mat-card-title><h2>Supplier ID: {{supplier.supplierId}}</h2></mat-card-title>
      <mat-card-subtitle>Company Name: {{supplier.companyName}}</mat-card-subtitle>
    </mat-card-header>
    <mat-card-content>
      <dl>
        <dt>Contact Name:</dt>
        <dd>{{supplier.contactName}}</dd>
        <dt>Contact Title:</dt>
        <dd>{{supplier.contactTitle}}</dd>
        <dt>Address:</dt>
        <dd>{{supplier.address}}</dd>
        <dt>City:</dt>
        <dd>{{supplier.city}}</dd>
        <dt>Region:</dt>
        <dd>{{supplier.region}}</dd>
        <dt>Postal Code:</dt>
        <dd>{{supplier.postalCode}}</dd>
        <dt>Country:</dt>
        <dd>{{supplier.country}}</dd>
        <dt>Phone:</dt>
        <dd>{{supplier.phone}}</dd>
        <dt>Fax:</dt>
        <dd>{{supplier.fax}}</dd>
        <dt>Home Page:</dt>
        <dd>{{supplier.homePage}}</dd>
      </dl>
    </mat-card-content>
    <mat-card-actions>
      <a mat-flat-button color="primary" [routerLink]="['/supplier-edit/', supplier.supplierId || 'all']"><
mat-icon>edit</mat-icon></a>
      <a mat-flat-button color="warn" (click)="deleteSupplier(supplier.supplierId)"><mat-icon>delete</mat-i
con></a>
    </mat-card-actions>
  </mat-card>
</div>
```

Finally, open and edit `src/app/supplier-detail/supplier-detail.component.css` then add these lines of CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-loading-shade {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 56px;
  right: 0;
  background: rgba(0, 0, 0, 0.15);
  z-index: 1;
  display: flex;
  align-items: center;
  justify-content: center;
}

.mat-flat-button {
  margin: 5px;
}
```

14. Add a Supplier using Angular 7 Material

To add a new supplier, we have to create Angular 7 reactive form. Open and edit `src/app/supplier-add/supplier-add.component.ts` then add this imports.

```
import { Router } from '@angular/router';
import { ApiService } from '../api.service';
import { FormControl, FormGroupDirective, FormBuilder, FormGroup, NgForm, Validators } from '@angular/form
s';
```

Inject above modules to the constructor.

```
constructor(private router: Router, private api: ApiService, private formBuilder: FormBuilder) { }
```

Declare variables for the Form Group and all of the required fields inside the form before the constructor.

```
supplierForm: FormGroup;
companyName = '';
contactName = '';
contactTitle = '';
address = '';
city = '';
region = '';
postalCode = '';
country = '';
phone = '';
fax = '';
homePage = '';
isLoadingResults = false;
```

Add initial validation for each field.

```
ngOnInit() {
  this.supplierForm = this.formBuilder.group({
    'companyName' : [null, Validators.required],
    'contactName' : [null, null],
    'contactTitle' : [null, null],
    'address' : [null, null],
    'city' : [null, null],
    'region' : [null, null],
    'postalCode' : [null, null],
    'country' : [null, null],
    'phone' : [null, null],
    'fax' : [null, null],
    'homePage' : [null, null]
  });
}
```

Create a function for submitting or POST supplier form.

```
onFormSubmit(form: NgForm) {
  this.isLoadingResults = true;
  this.api.addSupplier(form)
    .subscribe((res: { [x: string]: any; }) => {
        const supplier = res['supplierResponse'];
        const id = supplier['supplierId'];
        this.isLoadingResults = false;
        this.router.navigate(['/supplier-details', id]);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      });
}
```

Next, add this import for implementing `ErrorStateMatcher`.

```
import { ErrorStateMatcher } from '@angular/material/core';
```

Create a new class after the end of this class bracket.

```
/** Error when invalid control is dirty, touched, or submitted. */
export class MyErrorStateMatcher implements ErrorStateMatcher {
  isErrorState(control: FormControl | null, form: FormGroupDirective | NgForm | null): boolean {
    const isSubmitted = form && form.submitted;
    return !!(control && control.invalid && (control.dirty || control.touched || isSubmitted));
  }
}
```

Instantiate that `MyErrorStateMatcher` as a variable in main class.

```
matcher = new MyErrorStateMatcher();
```

Next, open and edit `src/app/supplier-add/supplier-add.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" [routerLink]="['/supplier']"><mat-icon>list</mat-icon></a>
  </div>
  <mat-card class="example-card">
    <form [formGroup]="supplierForm" (ngSubmit)="onFormSubmit(supplierForm.value)">
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Company Name" formControlName="companyName"
               [errorStateMatcher]="matcher">
        <mat-error>
          <span *ngIf="!supplierForm.get('companyName').valid && supplierForm.get('companyName').touched">P
lease enter Company Name</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Contact Name" formControlName="contactName"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Contact Title" formControlName="contactTitle"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Address" formControlName="address"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="City" formControlName="city"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Region" formControlName="region"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Postal Code" formControlName="postalCode"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Country" formControlName="country"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Phone" formControlName="phone"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Fax" formControlName="fax"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
```

```
        <input matInput placeholder="Home Page" formControlName="homePage"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <div class="button-row">
        <button type="submit" [disabled]="!supplierForm.valid" mat-flat-button color="primary"><mat-icon>sa
ve</mat-icon></button>
      </div>
    </form>
  </mat-card>
</div>
```

Finally, open and edit `src/app/supplier-add/supplier-add.component.css` then add this CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-form {
  min-width: 150px;
  max-width: 500px;
  width: 100%;
}

.example-full-width {
  width: 100%;
}

.example-full-width:nth-last-child() {
  margin-bottom: 10px;
}

.button-row {
  margin: 10px 0;
}

.mat-flat-button {
  margin: 5px;
}
```

15. Edit a Supplier using Angular 7 Material

We have put an edit button inside the Supplier Detail component for call Edit page. Now, open and edit `src/app/supplier-edit/supplier-edit.component.ts` then add these imports.

```
import { Router, ActivatedRoute } from '@angular/router';
import { ApiService } from '../api.service';
import { FormControl, FormGroupDirective, FormBuilder, FormGroup, NgForm, Validators } from '@angular/form
s';
import { ErrorStateMatcher } from '@angular/material/core';
```

Ad

Base solution for your next web application

VISIT SITE

Inject above modules to the constructor.

```
constructor(private router: Router, private route: ActivatedRoute, private api: ApiService, private formBui
lder: FormBuilder) { }
```

Declare the Form Group variable and all of the required variables for the supplier form before the constructor.

```
supplierForm: FormGroup;
companyName = '';
contactName = '';
contactTitle = '';
address = '';
city = '';
region = '';
postalCode = '';
country = '';
phone = '';
fax = '';
homePage = '';
isLoadingResults = false;
matcher = new MyErrorStateMatcher();
```

Next, add validation for all fields when the component is initiated.

```
ngOnInit() {
  this.getSupplier(this.route.snapshot.params['id']);
  this.supplierForm = this.formBuilder.group({
    'companyName' : [null, Validators.required],
    'contactName' : [null, null],
    'contactTitle' : [null, null],
    'address' : [null, null],
    'city' : [null, null],
    'region' : [null, null],
    'postalCode' : [null, null],
    'country' : [null, null],
    'phone' : [null, null],
    'fax' : [null, null],
    'homePage' : [null, null]
  });
}
```

Create a function for getting supplier data that filled to each form fields.

```
getSupplier(id: number) {
  this.api.getSupplier(id).subscribe(data => {
    this.supplierId = data.supplierId;
    this.supplierForm.setValue({
      companyName: data.companyName,
      contactName: data.contactName,
      contactTitle: data.contactTitle,
      address: data.address,
      city: data.city,
      region: data.region,
      postalCode: data.postalCode,
      country: data.country,
      phone: data.phone,
      fax: data.fax,
      homePage: data.homePage
    });
  });
}
```

Create a function to update the supplier changes.

```
onFormSubmit(form: NgForm) {
  this.isLoadingResults = true;
  this.api.updateSupplier(this.supplierId, form)
    .subscribe(res => {
        this.isLoadingResults = false;
        this.router.navigate(['/supplier']);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      }
    );
}
```

Add a function for handling show supplier details button.

```
supplierDetails() {
  this.router.navigate(['/supplier-details', this.supplierId]);
}
```

Add a class that implementing `ErrorStateMatcher` after the current class.

```
/** Error when invalid control is dirty, touched, or submitted. */
export class MyErrorStateMatcher implements ErrorStateMatcher {
  isErrorState(control: FormControl | null, form: FormGroupDirective | NgForm | null): boolean {
    const isSubmitted = form && form.submitted;
    return !!(control && control.invalid && (control.dirty || control.touched || isSubmitted));
  }
}
```

Next, open and edit `src/app/supplier-edit/supplier-edit.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" [routerLink]="['/supplier']"><mat-icon>list</mat-icon></a>
  </div>
  <mat-card class="example-card">
    <form [formGroup]="supplierForm" (ngSubmit)="onFormSubmit(supplierForm.value)">
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Company Name" formControlName="companyName"
               [errorStateMatcher]="matcher">
        <mat-error>
          <span *ngIf="!supplierForm.get('companyName').valid && supplierForm.get('companyName').touched">P
lease enter Company Name</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Contact Name" formControlName="contactName"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Contact Title" formControlName="contactTitle"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Address" formControlName="address"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="City" formControlName="city"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Region" formControlName="region"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Postal Code" formControlName="postalCode"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Country" formControlName="country"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Phone" formControlName="phone"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Fax" formControlName="fax"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <mat-form-field class="example-full-width">
```

```
        <input matInput placeholder="Home Page" formControlName="homePage"
               [errorStateMatcher]="matcher">
      </mat-form-field>
      <div class="button-row">
        <button type="submit" [disabled]="!supplierForm.valid" mat-flat-button color="primary"><mat-icon>up
date</mat-icon></button>
      </div>
    </form>
  </mat-card>
</div>
```

Finally, open and edit `src/app/supplier-edit/supplier-edit.component.css` then add this lines of CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-form {
  min-width: 150px;
  max-width: 500px;
  width: 100%;
}

.example-full-width {
  width: 100%;
}

.example-full-width:nth-last-child() {
  margin-bottom: 10px;
}

.button-row {
  margin: 10px 0;
}

.mat-flat-button {
  margin: 5px;
}
```

16. Run and Test the ASP.NET Core Web API and Angular 7 CRUD Web Application

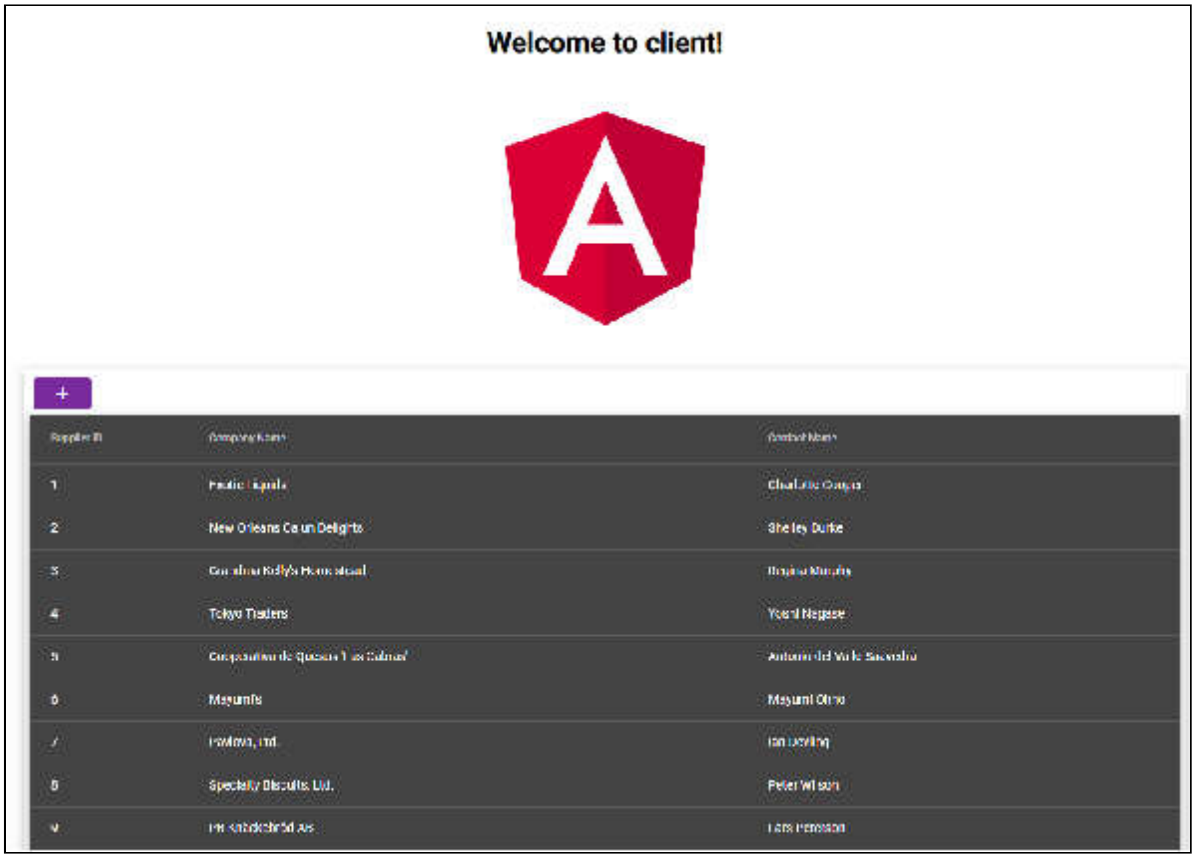
Before running the ASP.NET Core Web API that consumes by Angular 7 application, make sure ASP.NET Core Web API serve by `HTTP` only. Just open `Properties\\launchSettings.json` then remove `https` from the `applicationUrl`. Next, type this command in the Visual Studio Code Terminal.

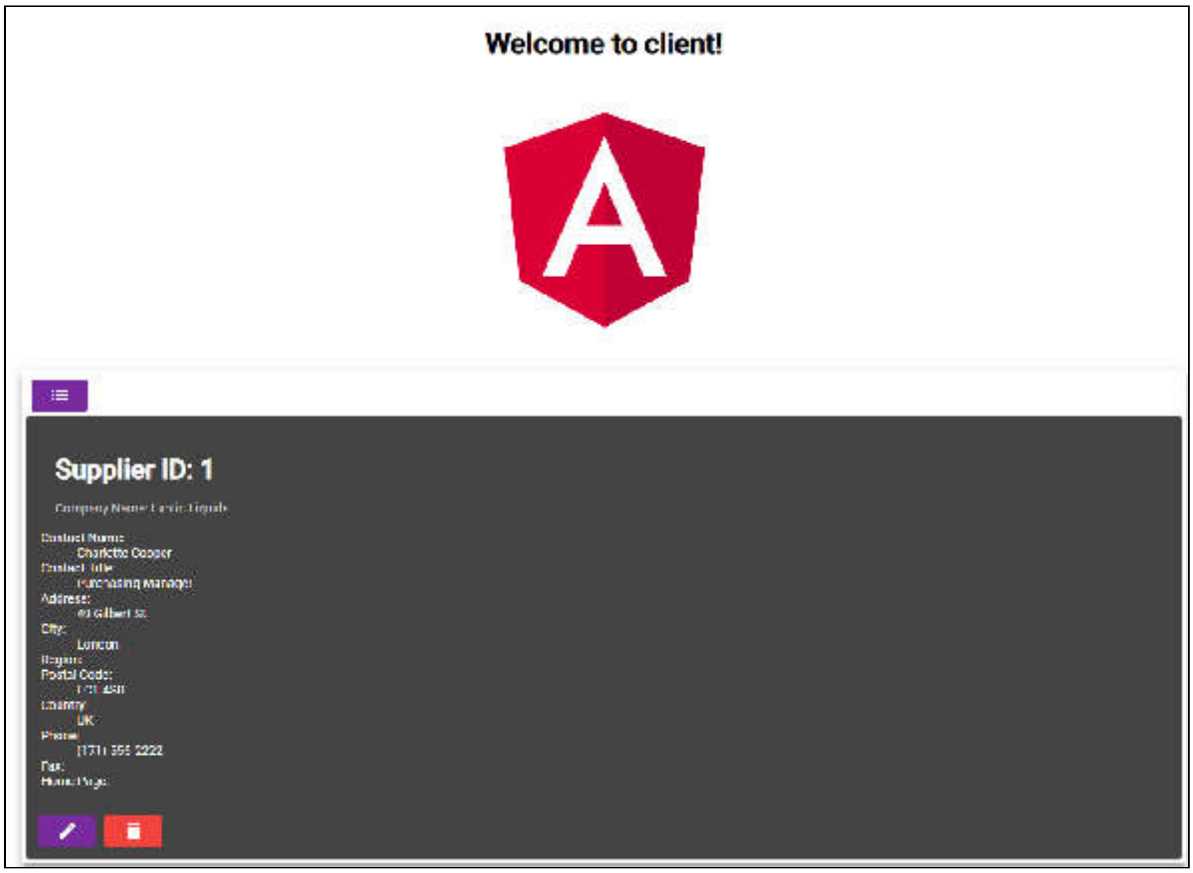
```
dotnet watch run
```

To run the Angular 7 application, type this command from the Node.js Command Prompt.

```
ng serve
```

Now, you will see this Angular 7 pages when pointing your browser to `localhost:4200`.





You can just navigate through the whole Angular 7 application.

That it's, the tutorial of Building Web App using ASP.NET Web API Angular 7 and SQL Server. You can find the fully working source code from our GitHub (\).

That just the basic. If you need more deep learning about ASP.NET Core, Angular or related you can take the following cheap course:

- Learning ASP.NET Web API (\)
- Beginning ASP.NET (\)
- Learn ASP.NET Core Step By Step (\)
- Learning Angular 7 (\)
- Angular 6, Angular 7 Step by Step for beginners (\)
- Angular 7 with Angular Material and Firebase Cloud Firestore (\)

Thanks!

Follow

(http://www.facebook.com/djamwarecom)

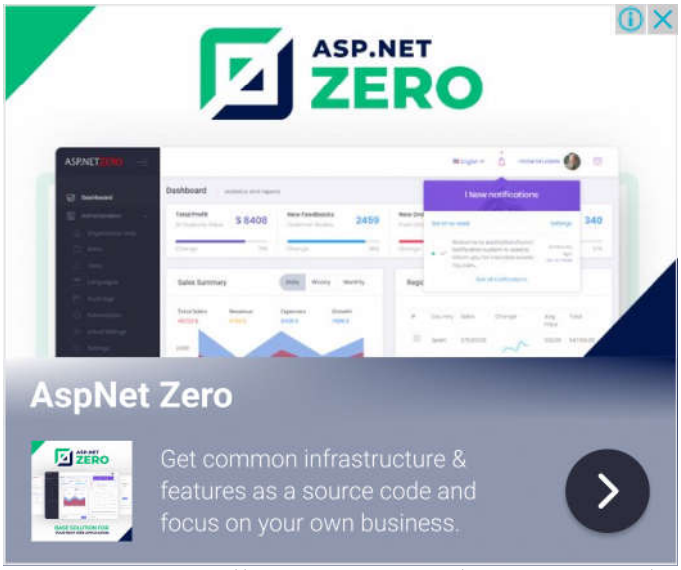
(http://twitter.com/intent/follow?source=followbutton&variant=1.0&screen_name=djamware)

(http://www.pinterest.com/djamware)

(http://www.linkedin.com/in/didin-jamaludin-7a530351)

(http://www.youtube.com/channel/UCtI81hYLh2Ae_45KHkyy0vw?sub_confirmation=1)

(https://github.com/didinj)



ezoic (https://www.ezoic.com/what-is-ezoic/)
report this ad

Next Article ➜
ASP Net Core, SQL Server, and Angular 7: Web App Authentication

(/post/5cc6624980aca754f7a9d1f3/asp-net-core-sql-server-and-angular-7-web-app-authentication)

Related Articles

- ASP Net Core, SQL Server, and Angular 7: Web App Authentication (/post/5cc6624980aca754f7a9d1f3/asp-net-core-sql-server-and-angular-7-web-app-authentication)

Sponsored Links

Amazingly Cheap Flights in Vietnam

TripsInsider

Play this Game for 1 Minute and see why everyone is addicted

Desert Order

Biggest Animals In Animal Kingdom

animalsfact

Aging Too Fast? Stop Eating These 7 Foods

Hearty Aging

Chiến thắng nỗi sợ nói tiếng Anh chỉ với 45 phút mỗi ngày

Topica Native VIP

20 Breathtaking Places to See Before You Die

ezzin.com

Disqus seems to be taking longer than usual. Reload?

Programming Blog
(/post-category/595f867edbd39e7571e183dc/programming-blog)

Ionic Framework
(/post-sub-category/5845691a80aca7763489d872/ionic-framework)

Java
(/post-sub-category/583d6c37fcbe614473a4c4e8/java)

Javascript
(/post-sub-category/583d6d30fcbe614473a4c4e9/javascript)

Node.js
(/post-sub-category/58a9196f80aca748640ce352/nodejs)

HTML 5 Tutorial
(/post-sub-category/584209dffcbe618f680bdc5c/html-5-tutorial)

Groovy and Grails
(/post-sub-category/585b3fa380aca73b19a2efd4/groovy-and-grails)

React Native
(/post-sub-category/5b4aa0b480aca707dd4f65a6/react-native)

CSS 3
(/post-sub-category/584249bde4d5d303658d1ecf/css-3)

ASP.NET Core
(/post-sub-category/5c50643780aca754f7a9d1e9/aspnet-core)

MongoDB
(/post-sub-category/5845677b80aca7763489d871/mongodb)

All Articles
(/public/allArticles)

Popular Articles:

- Angular 8 Tutorial: Learn to Build Angular 8 CRUD Web App (/post/5d0eda6f80aca754f7a9d1f5/angular-8-tutorial-learn-to-build-angular-8-crud-web-app)
- Angular 6 HttpClient: Consume RESTful API Example (/post/5b87894280aca74669894414/angular-6-httpclient-consume-restful-api-example)
- Ionic 4, Angular 7 and Cordova Tutorial: Build CRUD Mobile Apps (/post/5be52ce280aca72b942e31bc/ionic-4-angular-7-and-cordova-tutorial-build-crud-mobile-apps)
- Building Web App using ASP.NET Web API Angular 7 and SQL Server

(/post/5c50e5f280aca754f7a9d1eb/building-web-app-using-aspnet-web-api-angular-7-and-sql-server)

- Push Notification using Ionic 4 and Firebase Cloud Messaging (/post/5c6ccd1f80aca754f7a9d1ec/push-notification-using-ionic-4-and-firebase-cloud-messaging)
- ASP Net Core, SQL Server, and Angular 7: Web App Authentication (/post/5cc6624980aca754f7a9d1f3/asp-net-core-sql-server-and-angular-7-web-app-authentication)
- Ionic 4 and Angular 7 Tutorial: HTTP Interceptor Example (/post/5c42ca7580aca754f7a9d1e8/ionic-4-and-angular-7-tutorial-http-interceptor-example)
- Angular 6 Tutorial: Getting Started Build Angular 6 Web Application (/post/5afa31a780aca726dee1fd6c/angular-6-tutorial-getting-started-build-angular-6-web-application)
- Building CRUD Mobile App using Ionic 4, Angular 6 and Cordova (/post/5b5cffaf80aca707dd4f65aa/building-crud-mobile-app-using-ionic-4-angular-6-and-cordova)
- How to Upload File on Ionic 3 using Native File Transfer Plugin (/post/599da16580aca768e4d2b130/how-to-upload-file-on-ionic-3-using-native-file-transfer-plugin)

©2012-2018 Djamware.com | Privacy Policy (/public/privacy) | About Us (/public/about) | Contact Us (/public/contact) | RSS (https://www.djamware.com/feeds)