CRUD Operation With Angular 5 HTTP Client And ASP.NET Core Web API

Mangesh Gaherwar Updated date Jan 23 2018 99.9k 4 5

AngularWebAPI.zip | HttpClientDemo.zip

HTTP is the messaging system between the client and the server-client that sends the request and server responds with the proper message. Angular HTTP client is the toolkit which enables us to send and receive the data over the RESTful HTTP endpoints. In Angular 4.3, this Angular HTTP API was provided which in the extension to the existing API providing some new feature and added to its own package of the *@angular/common/http*

Let's divide this article into two sections - the first one being the Angular part and UI part while the second one is the Server-side code which will hold the API part of the project.

Client-side setup

To make the HTTP client module available in the application, we must make sure it is included and configured properly in the application. Let's see step by step how we can do this.

Import the HTTP client module in the application module or root module. So, our root module is app.module.ts.

```
import { HttpClientModule } from '@angular/common/http';
import {FormsModule} from '@angular/forms'
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,HttpClientModule,FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now, we have imported the HTTP Client into our application. We can use them in our component easily. For this demo, we are using the simple Employee as an entity and we are going to demo the Get, Post, Put and Delete Requests. For this demo purpose, let's add one component in our Angular application.

In my case, I have added the component Employee. The component looks like below.

```
<div class="container">
  <h3>Employee List</h3>
  <table class="table table-condensed">
    <thead>
      <tr>
        <td>    </td>
        <td>     </td>
        <td>     </td>
        <td>     </td>
        <td>  <a (click)="ShowRegForm(e)">Add New</a></td>
      </tr>
      <tr>
        <th>ID</th>
```

```
          <th>Email</th>
          <th>Edit</th>
        </tr>
      </thead>
      <tbody>
        <tr class="success" *ngFor="let e of employeelist ">
          <td> {{e.id}}</td>
          <td>{{e.fname}}</td>
          <td>{{e.lname}}</td>
          <td>{{e.email}}</td>
          <td><a (click)="ShowRegForm(e)">Edit</a></td>
          <td><a (click)="ShowRegFormForDelete(e)">Delete</a></td>
        </tr>
      </tbody>
    </table>
</div>
<hr >

<form #regForm="ngForm">
<div class="container" *ngIf="editCustomer">
  <h3>{{FormHeader}}</h3>
  <table class="table table-condensed">
    <tbody>
      <tr>
        <td>First Name</td>
        <td><input type="text" name="fname" [(ngModel)]='fname' ></td>
        </tr>
        <tr>
        <td>Last Name</td>
        <td><input type="text" name="lname" [(ngModel)]='lname'></td>
        </tr>
        <tr>
        <td> Email</td>
        <td><input type="text" name="email" [(ngModel)]='email'></td>
        </tr>
```

```
53.             <td><input type="button" value="Save" (click)="Save(regForm)"></td>
54.
55.         </tr>
56.       </tbody>
57.     </table>
58.   </div>
59. </form>
```

Code Description

Here, we have an HTML page which has provision to display the List of the employees present in the database and then the Option to Add, Edit, List and Delete

Here based on the Button input the Form Header will be set according to the operation such as delete, add and edit.

Next step is the component itself and we have done the code for that

Import Statements

```
01. import { Component, OnInit } from '@angular/core';
02. import {NgForm} from '@angular/forms'
03. import {FormsModule} from '@angular/forms'
04. import { Observable } from 'rxjs/Observable';
05. import 'rxjs/add/operator/do';
06. import 'rxjs/add/operator/filter';
07. import 'rxjs/add/operator/map';
08.
09. import {EmployeeDataService} from '../DataServices/EmployeeDataService';
10. import {employee} from '../Models/Employee';
```

These are the basic imports that we can see here out of them, the others being the basic imports. We have some interesting imports like do, filter and map -- these are the operators which are used to transform the result that we get from the service

the Data related operations?

Constructor and injecting the Data Service

Here we have used the Employee Data Service in our application to use it we have used the DI and injected it in constructor like below

```
constructor(private dataservice:EmployeeDataService) // Available at imports
    {

    }
```

Next section is the code which we are using to call the data services as below

```
ngOnInit()
    {
      this.dataservice.getEmployee().subscribe((tempdate) =>{  this.employeelist=tempdate;})
      ,err=>{
        console.log(err);
      }
    }
  ShowRegForm=function(employee)
  {
    this.editCustomer=true;
    if(employee!=null)
    {
      this.SetValuesForEdit(employee)

    }
    else{
      this.ResetValues();
    }
  }

  ShowRegFormForDelete=function(employee)
  {
```

```
      this.SetValuesForDelete(employee)

    }
  }

  SetValuesForDelete=function(employee)
{
  this.fname=employee.fname;
  this.lname=employee.lname;
  this.email=employee.email;
  this.id=employee.id;
  this.FormHeader="Delete"

}
//Function to set the values for edit form
SetValuesForEdit=function(employee)
{
  this.fname=employee.fname;
  this.lname=employee.lname;
  this.email=employee.email;
  this.id=employee.id;
  this.FormHeader="Edit"
}
//Function to reset the values
ResetValues(){
  this.fname="";
  this.lname="";
  this.email="";
  this.id="";
  this.FormHeader="Add"
}
//Common function for the Operation
  Save(regForm:NgForm)
  {
    this.GetDummyObject(regForm);
```

```
63.      {
64.      case "Add":
65.             this.Addemployee(this.Dummyemployee);
66.      break;
67.      case "Edit":
68.            this.UpdateEmployee(this.Dummyemployee);
69.      break;
70.      case "Delete":
71.            this.DeleteEmployee(this.Dummyemployee);
72.      break;
73.             default:
74.      break;
75.
76.      }
77.    }
78.
79.  GetDummyObject(regForm:NgForm):employee
80.  {
81.    this.Dummyemployee= new employee
82.    this.Dummyemployee.Email=regForm.value.email;
83.    this.Dummyemployee.Fname=regForm.value.fname;
84.    this.Dummyemployee.Lname=regForm.value.lname;
85.    this.Dummyemployee.ID=regForm.value.id;
86.    return this.Dummyemployee;
87.  }
88.    Addemployee(e: employee)
89.    {
90.      this.dataservice.AddEmployee(this.Dummyemployee).subscribe(res=>
91.        {
92.          this.employeelist.push(res);
93.          alert("Data added successfully !! ")
94.          this.editCustomer=false;
95.        })
96.        ,err=>
97.        {
```

```
100.    }
101.
102.    UpdateEmployee(e: employee)
103.    {
104.      this.dataservice.EditEmployee(this.Dummyemployee).subscribe(res=>
105.        {
106.            this.editCustomer=false;
107.            this.dataservice.getEmployee().subscribe(res=>{
108.            this.employeelist=res;
109.            });
110.            alert("Employee data Updated successfully !!")
111.        });
112.    }
113.
114.    DeleteEmployee(e: employee)
115.    {
116.      this.dataservice.DeleteEmployee(this.Dummyemployee).subscribe(res=>
117.        {
118.          this.editCustomer=false;
119.        this.dataservice.getEmployee().subscribe(res=>{
120.                    this.employeelist=res;
121.          });
122.          alert("employee Deleted successfully !! ")
123.        });
124.    }
```

We can see that we have called the get employee method from the ngOnInit Event of the component instead of calling in the constructor we have specifically done this to avoid the delay in loading of the component

Next, we have methods like Addemployee(),DeleteEmployee(),UpdateEmployee() which are used for the calling the Data service methods from the application like Add Edit and Delete Employees

Other methods are the supplementary methods which are used to clear the inputs and setting the object.

```
export const ROOT_URL:string="http://localhost:39029/api/";
```

Here in this code we have defined the Root_URL as the constant which holds the value of the API address. Next is the Model employee.ts which we use to map the Data which we are sending and receiving from the API code snippet for the same is

```
export interface employee{
    ID:string;
    Fname:string;
    Lname:string;
    Email:string;

}
```

Main and most important part of the application is the Data service which we have used

Code snippet for the same is as follows.

```
import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/retry';
import 'rxjs/add/observable/of';
import 'rxjs/Rx';

import {employee} from '../Models/Employee';
import {ROOT_URL} from '../Models/Config';
import { Injectable }   from '@angular/core';

@Injectable()
export class EmployeeDataService

{
    employees: Observable<employee[]>;
```

```
20.      constructor(private http:HttpClient)
21.      {
22.
23.      }
24.  getEmployee()
25.  {
26.   return this.http.get<employee[]>(ROOT_URL + '/Employees')
27.  }
28.  AddEmployee(emp:employee)
29.  {
30.
31.    const headers = new HttpHeaders().set('content-type', 'application/json');
32.    var body = {
33.                        Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email
34.                }
35.
36.  return this.http.post<employee>(ROOT_URL+'/Employees',body,{headers})
37.
38.  }
39.
40.  EditEmployee(emp:employee)
41.  {
42.      const params = new HttpParams().set('ID', emp.ID);
43.    const headers = new HttpHeaders().set('content-type', 'application/json');
44.    var body = {
45.                        Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email,ID:emp.ID
46.                }
47.          return this.http.put<employee>(ROOT_URL+'/Employees/'+emp.ID,body,{headers,params})
48.
49.  }
50.
51.
52.

53.  DeleteEmployee(emp:employee)
54.  {
```

```
  var body = {
                        Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email,ID:emp.ID
                }
          return this.http.delete<employee>(ROOT_URL+'/Employees/'+emp.ID)

}
}
```

This is the Data service class which we use to call the API. Let's see the methods and code description of the methods that are present in this class

To make the Http Client available in the class we have imported the http packages from common/http package which are as follows

```
Import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';
```

In this Http client have the http method like get, post, put etc.

HttpParams for sending the parameter to the methods like Put and delete

Another is the HttpHeaders which we can use to pass the Headers which can be used to pass the values

Next is as every call to the Http client methods returns the Observable so we need to get the Observable package and some Transform method as well like below,

```
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/retry';
import 'rxjs/add/observable/of';
import 'rxjs/Rx';
```

Next section is bringing the Employee model in application along with the App URL constant and @Injectable attribute so that we can inject this class as Dependency Injection to another class,

```
03. import { Injectable }    from '@angular/core';
```

After making the class injectable we have Added the Http Client service as a dependency in the application using Constructor in that we have created the private variable http which is of type HttpClient

First method we will see is the method getEmployee()

```
01. getEmployee()
02. {
03.  return this.http.get<employee[]>(ROOT_URL + '/Employees')
04. }
```

This method is returning the Observable of the employee after getting it from the API

One thing to notice here is that we are directly mapping the Response which we are getting from the API to the employee[] so Http Client allows us to map the response as a typed Response.

Next Method is the AddEmployee,

```
01. AddEmployee(emp:employee)
02. {
03.   const headers = new HttpHeaders().set('content-type', 'application/json');
04.   var data = {Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email}
05. return this.http.post<employee>(ROOT_URL+'/Employees', data,{headers})
06. }
```

This method accepts the employee object which is our model object and receiving from the component.

In this Method we have used the HttpHeaders to set the content type for the request which is application/json. Next, we have converted the data from the Employee object in the JSON and then passed to the Post method of the http which is the verb of the HTTP

Post method has a signature like

Another Operation is the EditEmployee which accepts the employee object as a parameter body of the method is as follows

```
EditEmployee(emp:employee)
{
    const params = new HttpParams().set('ID', emp.ID);
    const headers = new HttpHeaders().set('content-type', 'application/json');
  var body = {
                      Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email,ID:emp.ID
              }
```

return his.http.put<employee>(ROOT_URL+'/Employees/'+emp.ID,body,{headers,params})

In this method, we need the emp id as a parameter for the operation and we are passing it using the attribute HttpParams . we have set the Headers to the Application/json and converted the object to the json

Next is the call the Put method of the http and it accepts the URL along with the Body header and parameter .

Last method to complete our CRUD application is the Delete Method which have following structure

```
DeleteEmployee(emp:employee)
{
    const params = new HttpParams().set('ID', emp.ID);
  const headers = new HttpHeaders().set('content-type', 'application/json');
  var body = {Fname:emp.Fname,Lname:emp.Lname,Email:emp.Email,ID:emp.ID}
 return this.http.delete<employee>(ROOT_URL+'/Employees/'+emp.ID)
}
```

Here we again used the parameter for passing and for the delete the record we have set the headers and called the delete method of the http which will return the observable of the employee type and we can use it in the application

This was about the angular Code which we have used; let's see some web Api code which we will be using in our application .

rest of the stuff will demonstrate some basic settings we do to allow the angular app and avoid any CORS issues that may arise, for that we can check the Startup.cs which is as follows,

```
01. using System;
02. using System.Collections.Generic;
03. using System.Linq;
04. using System.Threading.Tasks;
05. using Microsoft.AspNetCore.Builder;
06. using Microsoft.AspNetCore.Hosting;
07. using Microsoft.Extensions.Configuration;
08. using Microsoft.Extensions.DependencyInjection;
09. using Microsoft.Extensions.Logging;
10. using Microsoft.Extensions.Options;
11. using Microsoft.EntityFrameworkCore;
12.
13. namespace AngularWebAPI
14. {
15.     public class Startup
16.     {
17.         public Startup(IConfiguration configuration)
18.         {
19.             Configuration = configuration;
20.         }
21.
22.         public IConfiguration Configuration { get; }
23.
24.         // This method gets called by the runtime. Use this method to add services to the container
25.         public void ConfigureServices(IServiceCollection services)
26.         {
27.             services.AddMvc();
28.             services.AddCors();
29.             services.AddDbContext<ApplicationDbContext>
    (options => options.UseSqlServer("Your Connection string"));
30.             services.AddCors(options =>
31.             {
```

```
                    .AllowAnyMethod()
                    .AllowAnyHeader()
                    .AllowCredentials());
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pi
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
        app.UseCorsMiddleware();
            app.UseMvc();
            app.UseCors("CorsPolicy");
        }
    }
}
```

In this startup we have added the MVC Entity Framework and other stuff to run the application along with we have added one policy for the CORS which will allow the request from any origin

Also we have added one middleware which will handle the incoming request. CORS issues code for the same can be like below

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

public class CorsMiddleware
{
    private readonly RequestDelegate _next;

    public CorsMiddleware(RequestDelegate next)
```

```
    }

    public Task Invoke(HttpContext httpContext)
    {
        httpContext.Response.Headers.Add("Access-Control-Allow-Origin", "*");
        httpContext.Response.Headers.Add("Access-Control-Allow-Credentials", "true");
        httpContext.Response.Headers.Add("Access-Control-Allow-Headers", "Content-
Type, Accept");
        httpContext.Response.Headers.Add("Access-Control-Allow-
Methods", "POST,GET,PUT,PATCH,DELETE,OPTIONS");
        return _next(httpContext);
    }
}

// Extension method used to add the middleware to the HTTP request pipeline.
public static class CorsMiddlewareExtensions
{
    public static IApplicationBuilder UseCorsMiddleware(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<CorsMiddleware>();
    }
}
```

This is middleware which adds the Headers in the incoming request and adds it in the Request pipeline.

In this API we are Using the Code First Approach of the Entity framework; here our model class will be employee which will be like below

```
namespace AngularWebAPI.Models
{
    public class Employee
    {
        public int ID { get; set; }
        public string Fname { get; set; }
        public string  Lname { get; set; }
```

```
}
```

Next is Adding the controller which will be a Web API controller and we can use this class to map all the operations and use them accordingly

```
namespace AngularWebAPI.Controllers
{
    [Produces("application/json")]
    [Route("api/Employees")]
    public class EmployeesController : Controller
    {
        private readonly ApplicationDbContext _context;

        public EmployeesController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/Employees
        [HttpGet]
        public IEnumerable<Employee> Getemployee()
        {
            return _context.employee;
        }

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

```

```
            if (employee == null)
            {
                return NotFound();
            }

            return Ok(employee);
        }

        // PUT: api/Employees/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutEmployee([FromRoute] int id, [FromBody] Employee employ
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (id != employee.ID)
            {
                return BadRequest();
            }

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

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

```
                }
            }

            return NoContent();
        }

        // POST: api/Employees
        [HttpPost]
        public async Task<IActionResult> PostEmployee([FromBody] Employee employee)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            _context.employee.Add(employee);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetEmployee", new { id = employee.ID }, employee);
        }

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

            var employee = await _context.employee.SingleOrDefaultAsync(m => m.ID == id);
            if (employee == null)
            {
                return NotFound();
            }
```

```
106.                 await _context.SaveChangesAsync();
107.
108.                 return Ok(employee);
109.             }
110.
111.             private bool EmployeeExists(int id)
112.             {
113.                 return _context.employee.Any(e => e.ID == id);
114.             }
115.         }
116.     }
```

This is the web api controller which we use for all the database operations, as we are using the database we can apply the migration and generate the table in the database.

Note

We can use a better approach for handling the database operation like repository and using DTO this is just for the explanation and to check how it works .

When we run the application we can see the output like below .


Asp.net Core

When we click the Add New it will Open this form


Asp.net Core

Same form will be used for the rest of the Operation.

This was about the using Http Client to read, add, delete and update the employee data in the database. We have used the Asp.net web API core for the Web API we have added the Middleware to avoid the CORS issues in this we have used the EF Code first approach for interacting with the database.

1. Angularcode
2. web API

References

- *https://blog.angularindepth.com/the-new-angular-httpclient-api-9e5c85fe3361*
- *https://medium.com/codingthesmartway-com-blog/angular-4-3-httpclient-accessing-rest-web-services-with-angular-2305b8fd654b*
- *https://Angular.io*

Next Recommended Article
CRUD Operations In ASP.NET Core Web API Using ADO.NET

Angular | Angular 5 HTTP Client | ASP.NET Core | ASP.NET Core web API | CRUD Operation

Mangesh Gaherwar TOP 500

217 622.3k 2

Hi I am Mangesh working as a Team Lead at the Intelegain technology since last 5 years,C# corner MVP ,love code ,books,Music and sharing the knowledge with the people

https://www.c-sharpcorner.com/members/mangesh-gaherwar2

5 4

Type your comment here and press Enter Key (Minimum 10 characters)

1876 16 0

1 2 Reply

What issues are you facing ??? Michael lemme know i will help you

Mangesh Gaherwar

217 9.2k 622.3k

Aug 10, 2018

0

There may be case that some thing have been changed during moderation better if u refer the git hub or use the downloaded code or follow the git hub link of the code it will be useful to u

Mangesh Gaherwar

217 9.2k 622.3k

Aug 10, 2018

0

Well written article.

Geshem Wanasinghe

1887 5 0

Jul 13, 2018

0 0 Reply