

RESTful Web Services - API

Vidly, curso ASPNET MVC 5 y Web API



Objetivo y descripción introductoria



El objetivo principal era crear una aplicación que cumpliera con una arquitectura **RESTful Web API**, incluyendo las funcionalidades principales de la misma.

Sin embargo, **noté que la implementación se estaba llevando en un entorno de desarrollo utilizando .NET Framework**, y por la diferencia de versiones, algunas sentencias, estructuras, librerías o funciones ya no cumplían o se encuentran discontinuadas, **por lo que decidí continuar con la implementación de la aplicación del curso utilizando .NET Core**.

El repositorio en cuestión, se encuentra creado, acondicionado y configurado en una arquitectura **ASP.NET Core Web API**.

De igual manera, este repositorio tiene como fin principal, demostrar el funcionamiento de las operaciones básicas de un servicio web REST, aún no se ha añadido funcionalidad más compleja a la aplicación.

Aspectos, conceptos y conocimientos aplicados



Debido a que la implementación fue realizada en ASP.NET Core, se realizaron algunos ajustes:

Dependency Injection

Algunos paquetes, librerías, namespaces y configuraciones fueron configuradas dentro del Startup.cs de la siguiente manera:

Configure Services

```
// This method gets called by the runtime. Use this method to add services to the container.
0 referencias
public void ConfigureServices(IServiceCollection services)
{
    services.AddAutoMapper(typeof(Startup)); // Configuración de clase Mapper vía Dependency Injection

    services.AddControllers().AddNewtonsoftJson(options =>
    {
        options.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
        options.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;
    });

    services.AddCors(c => c.AddPolicy("TestingPolicy", builder =>
    {
        builder.AllowAnyOrigin() // Configuración de políticas para aceptar o rechazar las solicitudes, metodos y orígenes de los distintos clientes.
        .AllowAnyMethod() // Por motivos de desarrollo y prueba, aceptará cualquier origen.
        .AllowAnyHeader();
    }));

    services.AddDbContext<TestingCoreContext>(options => // Configuración de DbContext para el mapeo de entidades y conexión con
    { // EntityFrameworkCore
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConenction"));
    });

    services.AddDefaultIdentity<ApplicationUser>() // Colocar clase/entidad de usuario por default para la autenticación de usuarios
    .AddEntityFrameworkStores<TestingCoreContext>(); // utilizando AspNetCore Identity
}
```

Configure

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseCors("TestingPolicy"); // Hacer el uso de las políticas declaradas en la configuración para aceptar o denegar las request.

    app.UseHttpsRedirection();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Controllers

Debido a que en esta plantilla o arquitectura de trabajo no se cuenta con vistas o está directamente configurada para funcionar como WebAPI, .NET Core hace más sencillo adaptar las rutas y la declaración de ApiControllers:

```

[ApiController]
[Route("api/[controller]")] Anotaciones y configuraciones para acceso al controlador
De esta forma pueden acceder con la URL localhost:9999/api/[controllername]
1 referencia
public class CustomersController : Controller
{
    private readonly TestingCoreContext _context;
    private readonly IMapper _mapper; Utilizar las dependencias inyectadas en la configuración global de la aplicación,
en este caso para utilizar los MappingProfiles y el DbContext
0 referencias
    public CustomersController(TestingCoreContext context, IMapper mapper)
    {
        this._context = context;
        this._mapper = mapper;
    }

    [HttpGet]
    0 referencias
    public IEnumerable<CustomerDTO> GetCustomers()
    {
        return _context.Customers.Include(c => c.MembershipType).ToList().Select(_mapper.Map<Customer, CustomerDTO>);
    }

    [HttpGet]
    [Route("{Id}")]
    1 referencia
    public IActionResult GetCustomer(int Id) {...}

    [HttpPost]
    0 referencias
    public IActionResult CreateCustomer(CustomerDTO customerDTO) {...}

    [HttpPut]
    [Route("{Id}")] Acostumbro a consumir los Ids de las request como parámetros dentro de la URL,
prefiero utilizar el body solo para consumir los datos completos que se envían en la
petición
    0 referencias
    public IActionResult UpdateCustomer(int Id, CustomerDTO customerDTO) {...}

    [HttpDelete]
    [Route("{Id}")]
    0 referencias
    public IActionResult DeleteCustomer(int Id) {...}
}

```

ActionResults

De igual forma y gracias al curso, he implementado las convenciones que mejor se adapten a una aplicación tipo RESTful, mandando los resultados correspondientes:

```

[HttpGet]
[Route("{Id}")]
1 referencia
public IActionResult GetCustomer(int Id)
{
    var customerInDb = _context.Customers.SingleOrDefault(c => c.Id == Id);

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

    return Ok(_mapper.Map<Customer, CustomerDTO>(customerInDb));
}

[HttpPost]
0 referencias
public IActionResult CreateCustomer(CustomerDTO customerDTO)
{
    if (!ModelState.IsValid)
    {
        return BadRequest();
    }

    Customer customer = _mapper.Map<CustomerDTO, Customer>(customerDTO);

    _context.Add(customer);
    _context.SaveChanges();

    customerDTO.Id = customer.Id;

    return CreatedAtAction(nameof(GetCustomer), new { Id = customer.Id }, customerDTO);

    //return Ok(customerDTO);
}

```

Siendo sincero y de manera muy personal, yo acostumbraba más a utilizar respuestas tipo **JsonResult** en la mayoría de mis respuestas, debido a que se me hacía más fácil adaptar las respuestas para los clientes de esta forma. Pero entiendo que, **al utilizar una WebAPI como medio de consumo para N cantidad y variedad de clientes, tiene que estar lo mejor adaptada para distintos escenarios.**

Funcionalidad



El proyecto contiene una funcionalidad basada en RESTful Web Services / API, por lo que se encarga de manejar todas las peticiones recibidas por los clientes, utilizando el protocolo HTTP como parte principal y sus métodos, los cuales son:

| Método | Función |
|--------|---|
| GET | Obtener, consultar, o leer información/recursos |
| POST | Envíar datos, información o recursos (normalmente para insertar datos en el servidor) |

| Método | Función |
|--------|---|
| PUT | Actualizar algún recurso del lado del servidor |
| DELETE | Eliminar/remove algún recurso u objeto del servidor |

Derivado de esto, la aplicación contiene las funciones para consultar, insertar, actualizar y eliminar registros que estén en una base de datos, siendo o actuando en su mayoría como un CRUD.

Pruebas



Las funcionalidades de esta aplicación han sido probadas en primera instancia utilizando la herramienta **Postman** para el manejo de solicitudes a los servicios y evaluación de las respuestas.

CustomersController

♦ GET Customers (api/customers)

The screenshot shows the Postman interface with a GET request to `https://localhost:44331/api/customers`. The response is a JSON array of four customer objects. The status is 200 OK, time is 995 ms, and size is 1.08 KB.

```

1  [
2    {
3      "id": 1,
4      "name": "Marie Williams",
5      "birthDate": null,
6      "isSubscribedInNewsletter": false,
7      "membershipTypeId": 1,
8      "membershipType": {
9        "id": 1,
10       "name": "Free"
11      },
12    },
13    {
14      "id": 3,
15      "name": "Juan Peralta",
16      "birthDate": "1982-08-15T00:00:00",
17      "isSubscribedInNewsletter": false,
18      "membershipTypeId": 2,
19      "membershipType": {
20        "id": 2,
21        "name": "Month"
22      },
23    },
24    {
25      "id": 4,
26      "name": "Elephant London"

```

GET Customer (api/customers/Id)

TestingCoreAPI / Customers / GET Customer

GET `https://localhost:44331/api/customers/3` [Send](#)

Params Authorization Headers (6) Body Pre-request Script Tests Settings [Cookies](#) [</>](#)

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (5) Test Results [Status: 200 OK](#) Time: 1041 ms Size: 383 B [Save Response](#)

Pretty Raw Preview Visualize JSON [Copy](#)

```
1 {
2   "id": 3,
3   "name": "Juan Peralta",
4   "birthDate": "1982-08-15T00:00:00",
5   "isSubscribedInNewsletter": false,
6   "membershipTypeId": 2,
7   "membershipType": {
8     "id": 2,
9     "name": "Month"
10  }
11 }
```

Runner 01:34 a. m. 06/05/2021

POST Customer (api/customer)

TestingCoreAPI / Customers / POST Customer

POST `https://localhost:44331/api/customers` [Petición POST en CustomersController](#) [Send](#)

Params Authorization Headers (8) Body Pre-request Script Tests Settings [Cookies](#) [</>](#)

none form-data x-www-form-urlencoded raw binary GraphQL JSON [Beautify](#)

```
1 {
2   "name": "Carmen Fernández",
3   "birthDate": "1982-08-15T00:00:00",
4   "isSubscribedInNewsletter": true,
5   "membershipTypeId": 3
6 }
```

Body Cookies Headers (6) Test Results [Status: 201 Created](#) Time: 184 ms Size: 409 B [Save Response](#)

Pretty Raw Preview Visualize JSON [Copy](#)

```
1 {
2   "id": 5,
3   "name": "Carmen Fernández",
4   "birthDate": "1982-08-15T00:00:00",
5   "isSubscribedInNewsletter": true,
6   "membershipTypeId": null,
7   "membershipType": {
8     "id": 3
9   }
10 }
```

Retorna objeto insertado en la base de datos con éxito

Devolución de respuesta y código de estado 201

Runner 01:03 a. m. 06/05/2021

♦ PUT Customer (api/customer/Id)

The screenshot shows the Postman interface for a PUT request to `https://localhost:44331/api/customers/2`. The request body is a JSON object with the following fields: `name` (Marco Tellez), `birthDate` (1992-11-21T00:00:00), `isSubscribedInNewsletter` (false), and `membershipTypeId` (4). The response status is 200 OK, with a time of 34 ms and a size of 259 B. The response body is a plain text message: "The record was updated successfully!".

TestingCoreAPI / Customers / PUT Customer

PUT `https://localhost:44331/api/customers/2`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "Marco Tellez",
3   "birthDate": "1992-11-21T00:00:00",
4   "isSubscribedInNewsletter": false,
5   "membershipTypeId": 4
6 }
```

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 34 ms Size: 259 B Save Response

Pretty Raw Preview Visualize Text

```
1 The record was updated successfully!
```

The screenshot shows the Postman interface for a GET request to `https://localhost:44331/api/customers`. The response status is 200 OK, with a time of 10 ms and a size of 1.11 KB. The response body is a JSON array of customer objects. A blue box highlights the first object, which matches the data from the previous PUT request. A note next to the box says "Registro actualizado con éxito".

GET `https://localhost:44331/api/customers`

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 10 ms Size: 1.11 KB Save Response

Pretty Raw Preview Visualize JSON

```
7 {
8   "membershipTypeId": null,
9   "membershipTypeId": 1
10 },
11 {
12   "id": 2,
13   "name": "Marco Tellez",
14   "birthDate": "1992-11-21T00:00:00",
15   "isSubscribedInNewsletter": false,
16   "membershipTypeId": null,
17   "membershipTypeId": 4
18 },
19 {
20   "id": 3,
21   "name": "Juan Peralta",
22   "birthDate": "1982-08-15T00:00:00",
23   "isSubscribedInNewsletter": false,
```

Registro actualizado con éxito

DELETE Customer (api/delete/Id)

The screenshot shows the Postman interface for a DELETE request. The request is configured with the method **DELETE** and the URL `https://localhost:44331/api/customers/2`. The response is displayed in the **Body** tab, showing a status of **200 OK** with a message: `1 The record was deleted successfully!`.

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Response

Status: 200 OK Time: 29 ms Size: 366 B

1 The record was deleted successfully!

MoviesController

GET Movies (api/movies)

The screenshot shows the Postman interface for the GET /api/movies endpoint. The response is a JSON array of movie objects, with the first object highlighted by a blue box:

```
{
  "id": 1,
  "name": "Shrek",
  "numberInStock": 5,
  "releaseDate": "2002-04-21T00:00:00",
  "dateAdded": "2021-05-05T00:00:00",
  "genre": {
    "id": 1,
    "name": "Comedy"
  },
  "genreId": 1
},
{
  "id": 2,
  "name": "Spiderman",
  "numberInStock": 4,
  "releaseDate": "2001-09-10T00:00:00",
  "dateAdded": "2021-01-01T00:00:00",
  "genre": {
    "id": 2,
    "name": "Action"
  },
  "genreId": 2
},
{
  "id": 3,
```

The status bar indicates a 200 OK response with a time of 988 ms and a size of 112 KB.

GET Movie (api/movies/Id)

The screenshot shows the Postman interface for the GET /api/movies/2 endpoint. The response is a JSON object representing the movie Spiderman:

```
{
  "id": 2,
  "name": "Spiderman",
  "numberInStock": 4,
  "releaseDate": "2001-09-10T00:00:00",
  "dateAdded": "2021-01-01T00:00:00",
  "genre": {
    "id": 2,
    "name": "Action"
  },
  "genreId": 2
}
```

The status bar indicates a 200 OK response with a time of 102 ms and a size of 390 B.

POST Movie (api/movies)

The screenshot shows the Postman application interface. At the top, there's a search bar and navigation icons. Below, the 'Movies' collection is selected, and the 'POST POST Movie' request is highlighted. The request is configured with the URL 'https://localhost:44331/api/movies' and the method 'POST'. The 'Body' tab is active, showing a JSON payload:

```
{  "name": "Rocky",  "numberInStock": 9,  "releaseDate": "1982-10-02T00:00:00",  "dateAdded": "2021-02-01T00:00:00",  "genreId": 2}
```

. The status bar at the bottom indicates a successful response: 'Status: 201 Created', 'Time: 173 ms', 'Size: 401 B'. The response body is displayed in the 'Body' tab, showing a JSON object:

```
{  "id": 5,  "name": "Rocky",  "numberInStock": 9,  "releaseDate": "1982-10-02T00:00:00",  "dateAdded": "2021-02-01T00:00:00",  "genre": null,  "genreId": 2}
```

. The Windows taskbar at the bottom shows the time as 01:47 a.m. on 06/05/2021.

PUT Movie (api/movies/Id)

The screenshot shows the Postman application interface. The 'PUT PUT Movie' request is highlighted, with the URL 'https://localhost:44331/api/movies/2' and method 'PUT'. The 'Body' tab is active, showing a JSON payload:

```
{  "name": "Terminator",  "numberInStock": 7,  "releaseDate": "1991-07-05T00:00:00",  "dateAdded": "2021-02-01T00:00:00",  "genreId": 2}
```

. The status bar at the bottom indicates a successful response: 'Status: 200 OK', 'Time: 35 ms', 'Size: 259 B'. The response body is displayed in the 'Body' tab, showing a text message: '1 The record was updated successfully!'. The Windows taskbar at the bottom shows the time as 01:49 a.m. on 06/05/2021.

Postman interface showing a GET request to `https://localhost:44331/api/movies`. The response is a JSON array of movie objects, with the second object (Terminator) highlighted by a blue box.

```
4 {
5   "name": "Shrek",
6   "numberInStock": 5,
7   "releaseDate": "2002-04-21T00:00:00",
8   "dateAdded": "2021-05-05T00:00:00",
9   "genre": {
10     "id": 1,
11     "name": "Comedy"
12   },
13   "genreId": 1
14 }
15 {
16   "id": 2,
17   "name": "Terminator",
18   "numberInStock": 7,
19   "releaseDate": "1991-07-05T00:00:00",
20   "dateAdded": "2021-02-01T00:00:00",
21   "genre": {
22     "id": 2,
23     "name": "Action"
24   },
25   "genreId": 2
26 }
27 {
28   "id": 3,
29   "name": "Star Wars Episode I",
30   "numberInStock": 3,
31   "releaseDate": "1997-10-01T00:00:00"
32 }
```

DELETE Movie (api/movies/Id)

Postman interface showing a DELETE request to `https://localhost:44331/api/movies/3`. The response is a text message: "The record was deleted successfully!"

```
1 The record was deleted successfully!
```

Pruebas con otros Frameworks



De igual manera, he decidido realizar un pequeño proyecto/aplicación en angular para manejar el apartado del cliente, y así, poder realizar las consultas y peticiones a la Web API.

Adjunto una pequeña demostración en donde nuestra aplicación de angular hace un llamado a nuestra Web API mandando una solicitud GET al controlador CustomersController:

Vista Customer en AngularJS

| Name | Birth Date | Membership Type | Options |
|-----------------|------------|-----------------|---------|
| Maria Williams | | Free | Delete |
| Juan Peralta | 1982-08-15 | Month | Delete |
| Kirsten London | 1972-09-10 | Year | Delete |
| Teresa Gonzalez | 1967-08-23 | Year | Delete |
| Adrian Nava | 1987-08-25 | Free | Delete |
| Luis Silva | 1997-11-21 | Free | Delete |

```
AngularJS: [Warning] $digest() reached $root without $pending, so $digest() will be performed again.
customers.component.ts:15
(6) ([-], [-], [-], [-], [-], [-])
  0: {id: 1, name: "Maria Williams", birthDate: null, isSubscribedInfo: true}
  1: {id: 3, name: "Juan Peralta", birthDate: "1982-08-15T00:00:00", isSubscribedInfo: true}
  2: {id: 4, name: "Kirsten London", birthDate: "1972-09-10T00:00:00", isSubscribedInfo: true}
  3: {id: 5, name: "Teresa Gonzalez", birthDate: "1967-08-23T00:00:00", isSubscribedInfo: true}
  4: {id: 6, name: "Adrian Nava", birthDate: "1987-08-25T00:00:00", isSubscribedInfo: true}
  5: {id: 8, name: "Luis Silva", birthDate: "1997-11-21T00:00:00", isSubscribedInfo: true}
  length: 6
  __proto__: Array(0)
[MD5] Live Reloading enabled. client:152
```

Componente Customer en AngularJS

```

@Component({
  selector: 'app-customers',
  templateUrl: './customers.component.html',
  styleUrls: ['./customers.component.css']
})
export class CustomersComponent implements OnInit {

  constructor(private service: SharedService, private datePipe: DatePipe) { }

  CustomerList: any = [];

  oCustomer: any;

  ngOnInit(): void {
    this.refreshCustomerList();
  }

  refreshCustomerList() {
    this.service.getCustomers().subscribe(data => {
      this.CustomerList = data;
      console.log(data);
    });
  }

  deleteCustomer(customer) {
    console.log(customer);
    this.service.deleteCustomer(customer.id).subscribe(data => {
      console.log(data);
    });
  }
}

```

Obtener la lista de clientes de nuestra WebAPI

```

1
2 <div>
3   <h1>Customers</h1>
4 </div>
5
6 <div>
7   <table class="table table-hover">
8     <thead class="thead-dark">
9       <tr>
10        <th>Name</th>
11        <th>Birth Date</th>
12        <th>Membership Type</th>
13        <th>Options</th>
14      </tr>
15    </thead>
16    <tbody>
17      <tr *ngFor="let customer of CustomerList">
18        <td>{{customer.name}}</td>
19        <td>{{customer.birthDate | date: 'yyyy-MM-dd'}}</td>
20        <td>{{customer.membershipType.name}}</td>
21        <td>
22          <button class="btn btn-danger" (click)="deleteCustomer(customer)">
23            Delete
24          </button>
25        </td>
26      </tr>
27    </tbody>
28  </table>
29 </div>

```

Colección obtenida de nuestra Web API

Parte técnica y declaración de la URL en Angular

Creé un servicio el cual tiene declarada la URL de la API desarrollada en ASP.NET Core, de esta manera, solo es ir agregando los métodos que tenga el API para poder realizar las diferentes acciones:

SharedService AngularJS

The screenshot shows a Visual Studio Code editor with a TypeScript file named `shared.service.ts`. The code defines an `Injectable` class `SharedService` with a `readonly APIUrl` property and several methods: `getCustomers`, `getCustomer`, `postCustomer`, `putCustomer`, and `deleteCustomer`. Annotations highlight the `APIUrl` declaration and the methods. Below the editor, a terminal window shows build logs for two chunks, indicating successful compilation and bundle generation.

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class SharedService {
9   readonly APIUrl = "https://localhost:44331/api";
10
11   constructor(private http: HttpClient) { }
12
13   /* Customers */
14   getCustomers(): Observable<any[]> {
15     return this.http.get<any>(this.APIUrl + '/customers');
16   }
17
18   getCustomer(Id: any): any {
19     return this.http.get<any>(this.APIUrl + '/customers/' + Id);
20   }
21
22   postCustomer(oCustomer: any) {
23     return this.http.post(this.APIUrl + '/customers', oCustomer);
24   }
25
26   putCustomer(Id: any, oCustomer: any) {
27     return this.http.put(this.APIUrl + '/customers/' + Id, oCustomer);
28   }
29
30   deleteCustomer(Id: any) {
31     return this.http.delete(this.APIUrl + '/customers/' + Id);
32   }
33 }
```

Terminal output:

```
4 unchanged chunks
Build at: 2021-05-06T07:35:41.134Z - Hash: 69f6d649fd49372e86cf - Time: 266ms
✓ Compiled successfully.
✓ Browser application bundle generation complete.
Initial Chunk Files | Names | Size
main.js | main | 27.99 kB
4 unchanged chunks
Build at: 2021-05-06T07:35:57.535Z - Hash: 9af15761181fa1d1918f - Time: 172ms
✓ Compiled successfully.
```

Notas Generales/Conclusiones



Aprovecho este espacio para poder comentar mis conclusiones y algunas notas extra.

El objetivo de este proyecto/aplicación era demostrar las funciones básicas/principales de una aplicación RESTful, la cuál normalmente se implementa del lado del servidor y que está orientada a resolver peticiones por parte de los clientes manejando como parte base el protocolo HTTP y todas sus acciones/métodos.

Debido a que era un proyecto de introducción, **aun falta mayor funcionalidad respecto al curso.**

Algunos de esos conceptos faltantes era la implementación de autenticación de usuario, roles y otros aspectos utilizando Identity.

Planeo aplicarlos en este proyecto debido a que. **he escuchado y he visto distintas formas de validar las sesiones y la autenticación, tal como el caso de JWT (JsonWebToken) entre otros.**

Veo que el manejo de un ORM es de bastante ayuda debido a que, al refactorizar o realizar ajustes mayores dentro de una WebAPI, expone menos a los clientes que la consumen, ya que, si son ajustes no controlados, impactaría a todos los clientes enlazados a la misma.

Realmente espero que este trabajo haya sido de su agrado y haya demostrado conceptos que he aprendido tanto de forma autodidáctica, y con la ayuda del curso que me compartieron.

Pienso que, uno de los obstáculos de este análisis, estudio e incluso implementación, fue el traspasar ciertos conocimientos en una arquitectura .NET Framework a .NET Core.