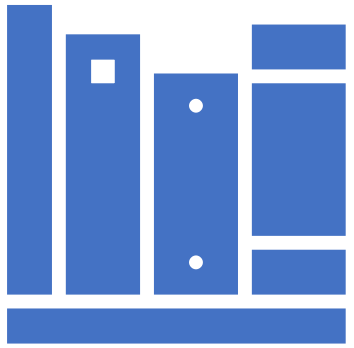


Arctech Info Private Limited

REST APIs with .NET and C#

REST APIs with .NET and C#

- ASP.NET makes it easy to build services that reach a broad range of clients, including browsers and mobile devices.
- With ASP.NET you use the same framework and patterns to build both web pages and services, side-by-side in the same project.

Simple serialization

```
[ApiController]
public class PeopleController : ControllerBase
{
    [HttpGet("people/all")]
    public ActionResult<IEnumerable<Person>> GetAll()
    {
        return new []
        {
            new Person { Name = “Raman" },
            new Person { Name = “Kappan" },
            new Person { Name = “Sheela" }
        };
    }
}

public class Person
{
    public string Name { get; set; }
}
```

- ASP.NET was designed for modern web experiences.
- Endpoints automatically serialize your classes to properly formatted JSON out of the box.
- No special configuration is required.
- Of course, serialization can be customized for endpoints that have unique requirement

```
https://localhost:5001/people/all

[{"name":"Ana"},{"name":"Felipe"},{"name":"Emillia"}]
```

Authentication and authorization

- Secure API endpoints with built-in support for industry standard JSON Web Tokens (JWT).
  Policy-based authorization gives you the flexibility to define powerful access control rules—all in code.

Routing alongside your code

- ASP.NET lets you define routes and verbs inline with your code, using attributes.
- Data from the request path, query string, and request body are automatically bound to method parameters

```
[ApiController]
public class PeopleController : ControllerBase
{
    [HttpGet("people/all")]
    public ActionResult<IEnumerable<Person>> GetAll()
    {
        return new []
        {
            new Person { Name = “Raman" },
            new Person { Name = “Kappan" },
            new Person { Name = “Sheela" }
        };
    }

    [HttpPost("people/create")]
    public IActionResult Create(Person person)
    {
        db.Add(person);
        db.SaveChanges();
        return Accepted();
    }
}
```

Features of Asp.NET Core web API

- The ASP.NET Core MVC controllers and web API controllers are unified
- Ability to develop and run-on Windows, macOS, and Linux.
- Open-source and community-focused.
- Built-in dependency injection.
- A lightweight, high-performance, and modular HTTP request pipeline.
- Ability to host on Kestrel, IIS, HTTP.sys, Nginx, Apache, and Docker.
- Side-by-side versioning.
- Tooling that simplifies modern web development.

Web API rules to bind parameters

- If the parameter is a "simple" type, Web API tries to get the value from the URI.
  - Simple types include the .NET primitive types (int, bool, double, and so forth)
  - TimeSpan, DateTime, Guid, decimal, and string,
  - Any type with a type converter that can convert from a string.
- For complex types, Web API tries to read the value from the message body, using a media-type formatter.

Using [FromUri]

- To force Web API to read a complex type from the URI, add the [FromUri] attribute to the parameter

```
public class GeoPoint
{
    public double Latitude { get; set; }
    public double Longitude { get; set; }
}
[ApiController]
public ValuesController : ControllerBase
{
    public IActionResult Get([FromUri] GeoPoint location)
    {
         ...
    }
}

http://localhost/api/values?Latitude=47.678558&Longitude=-122.130989
```

Using [FromBody]

- To force Web API to read a simple type from the request body, add the [FromBody] attribute to the parameter:

```
[ApiController]
public ValuesController : ControllerBase
{
    public IActionResult Post([FromBody] string name)
    {
        ...
    }
}
```

```
POST http://localhost:5076/api/values HTTP/1.1
User-Agent: Fiddler
Host: localhost:5076
Content-Type: application/json
Content-Length: 7

"Alice"
```

Controller action return types

- ASP.NET Core offers the following options for web API controller action return types:
  - Specific type (avoid using)
    - string or a custom object type

      ```
      [HttpGet]
      public List<Product> Get() { return _repository.GetProducts(); }
      ```
    - IEnumerable<T> or IAsyncEnumerable<T>
  - IActionResult
    - Is the base interface of multiple ActionResult classes
    - ActionResults represent various HTTP response codes
  - ActionResult<T>
    - An action with this return type can return HTTP response codes or custom data objects
    - When data objects are returned the response automatically includes HTTP OK (200)

Return examples

- `public async Task<ActionResult<IEnumerable<Hotel>>> GetHotels()`
  - `return await _context.Hotels.ToListAsync();`
- `public async Task<ActionResult<Hotel>> GetHotel(int id)`
  - `return NotFound();`
  - `return hotel;`
- `public async Task<IActionResult> PutHotel(int id, Hotel hotel)`
  - `return BadRequest();`
  - `return NoContent();`
- `public async Task<IActionResult> DeleteHotel(int id)`
  - `return NotFound();`
  - `return NoContent();`

HTTP methods

Action	HTTP method	Relative URI example
Get All products	GET	/api/products
Get a product by ID	GET	/api/products/id
Create a new product	POST	/api/products
Update a product	PUT	/api/products/id
Delete a product	DELETE	/api/products/id

Consuming a Web API

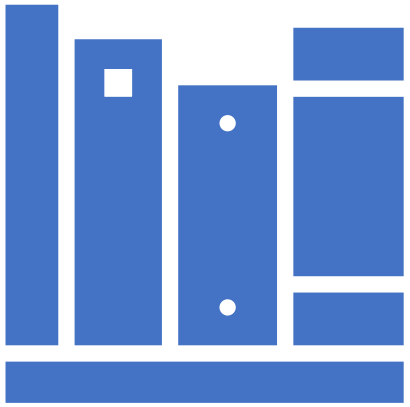
- See Example

Data Transfer Objects in Web API

- In the example, the web API exposes the database entities to the client.
- The client receives data that maps directly to your database tables.
- However, that's not always a good idea.
- Sometimes you want to change the shape of the data that you send to client. For example, you might want to
  - Hide particular properties that clients are not supposed to view.
  - Omit some properties in order to reduce payload size.
  - Flatten object graphs that contain nested objects, to make them more convenient for clients.
  - Decouple your service layer from your database layer.

DTO & their importance

- A DTO is an object that defines how the data will be sent over the network.
- It is a POCO that maps to the Data object
- A DTO is helpful whenever you need to group values in ad hoc structures for passing data around.
- DTOs help to further decouple presentation from the service layer and the domain model
- When DTOs are used, the presentation layer and the service layer share data contracts rather than classes.




Thank You

Avinash Tauro – Arctech Info Private Limited