<https://docs.microsoft.com/en-us/learn/modules/build-web-api-aspnet-core/>

# REST in ASP.NET Core

When you browse to a webpage, the web server communicates with your browser by using HTML, CSS, and JavaScript. If you interact with the page by doing something like submitting a sign-in form or selecting a buy button, the browser sends the information back to the web server.

In a similar way, web servers can communicate with a broad range of clients (browsers, mobile devices, other web servers, and more) by using web services. API clients communicate with the server over HTTP, and the two exchange information by using a data format such as JSON or XML. APIs are often used in single-page applications (SPAs) that perform most of the user interface logic in a web browser. Communication with the web server primarily happens through web APIs.

## REST: A common pattern for building APIs with HTTP

Representational State Transfer (REST) is an architectural style for building web services. REST requests are made over HTTP. They use the same HTTP verbs that web browsers use to retrieve webpages and to send data to servers. The verbs are:

* GET: Retrieve data from the web service.
* POST: Create a new item of data on the web service.
* PUT: Update an item of data on the web service.
* PATCH: Update an item of data on the web service by describing a set of instructions about how the item should be modified. The sample application in this module doesn't use this verb.
* DELETE: Delete an item of data on the web service.

Web service APIs that adhere to REST are called RESTful APIs. They're defined through:

* A base URI.
* HTTP methods, such as GET, POST, PUT, PATCH, or DELETE.
* A media type for the data, such as JavaScript Object Notation (JSON) or XML.

An API often needs to provide services for a few different but related things. For example, our pizza API might manage pizzas, customers, and orders. We use routing to map URIs to logical divisions in our code, so that requests to https://localhost:5000/pizza are routed to PizzaController. Requests to https://localhost:5000/order are routed to OrderController.

## Benefits of creating APIs in ASP.NET Core

With ASP.NET, you can use the same framework and patterns to build both webpages and services. This means you can reuse model classes and validation logic, and even serve both webpages and services side by side in the same project. This approach has benefits:

* **Simple serialization**: 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, you can [customize serialization](https://docs.microsoft.com/en-us/aspnet/core/web-api/advanced/custom-formatters) for endpoints that have unique requirements.
* **Authentication and authorization**: For security, API endpoints have built-in support for industry-standard JSON Web Tokens (JWTs). Policy-based authorization gives you the flexibility to define powerful access control rules in code.
* **Routing alongside your code**: ASP.NET lets you define routes and verbs inline with your code, by using attributes. Data from the request path, query string, and request body are automatically bound to method parameters.
* **HTTPS by default**: HTTPS is an important part of modern, professional web APIs. It relies on end-to-end encryption to provide privacy and to help ensure that your API calls aren't intercepted and altered between client and server.

ASP.NET provides support for HTTPS out of the box. It automatically generates a test certificate and easily imports it to enable local HTTPS. So you run and debug your applications securely, before you publish them.

### Share code and knowledge with .NET apps

Use your .NET skills and ecosystem to share logic from your web API to other apps built with .NET, including mobile, web, desktop, and services.

## Testing web APIs by using the .NET HTTP REPL

When you're developing a traditional website, you usually view and test your work in a web browser. Web APIs accept and return data rather than HTML, so a web browser isn't the best web API testing tool.

One of the easiest options for exploring and interacting with web APIs is the .NET HTTP REPL. REPL stands for read-evaluate-print loop. It's a simple and popular way to a build interactive command-line environments. In the next unit, you'll create a simple web API and then interact with it by using the .NET HTTP REPL.