

Web APIs

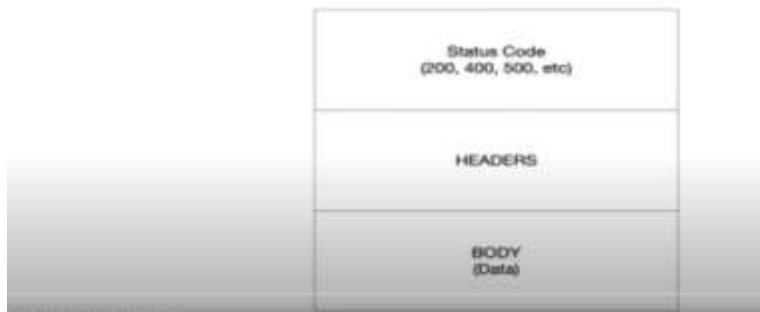
Traditional web applications handle requests by returning HTML to the user, which is displayed in a web browser. You can easily build applications of this nature using Razor Pages to generate HTML with Razor templates. This approach is common and well understood, but the modern application developer has a number of other possibilities to consider. Client-side single-page applications (SPAs) have become popular in recent years with the development of frameworks such as Angular, React, and Vue. These frameworks typically use JavaScript that runs in a user's web browser to generate the HTML they see and interact with. The server sends this initial JavaScript to the browser when the user first reaches the app. The user's browser loads the JavaScript and initializes the SPA before loading any application data from the server.

The only thing that differentiates MVC and API from a code perspective is the type of data they return—MVC controllers typically return a `ViewResult`; Web API controllers generally return raw .NET objects from their action methods, or an `IActionResult` such as `StatusCodeResult`.

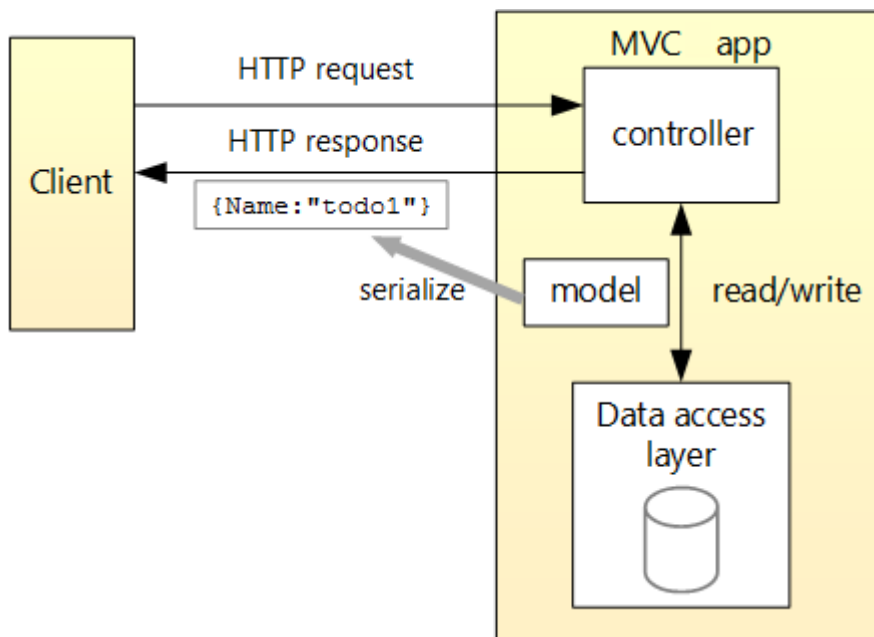
HTTP Request



HTTP Response



The diagram shows the design of the modern app



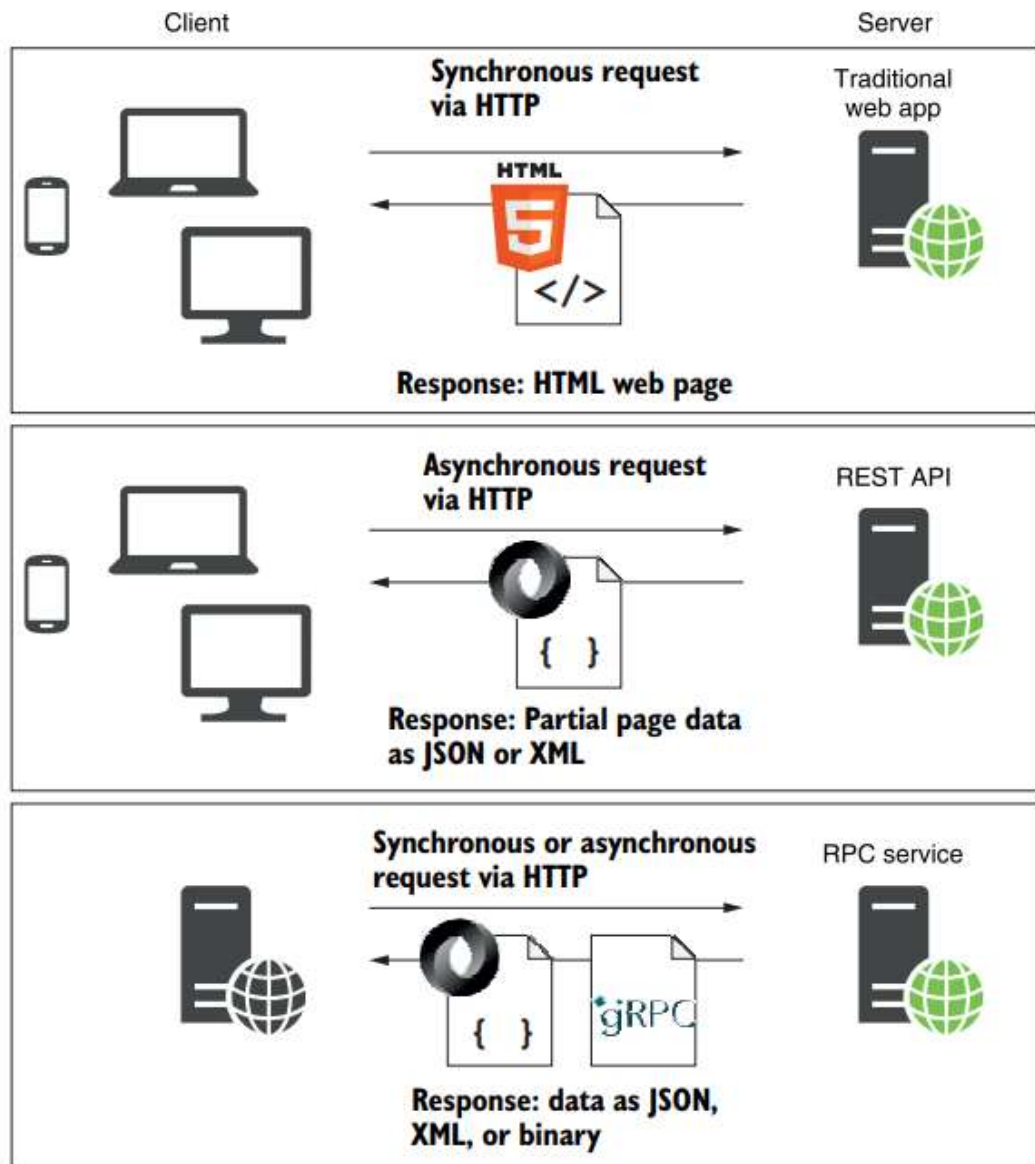


Figure 9.1 Modern developers have to consider a number of different consumers of their applications. As well as traditional users with web browsers, these could be SPAs, mobile applications, or other apps.

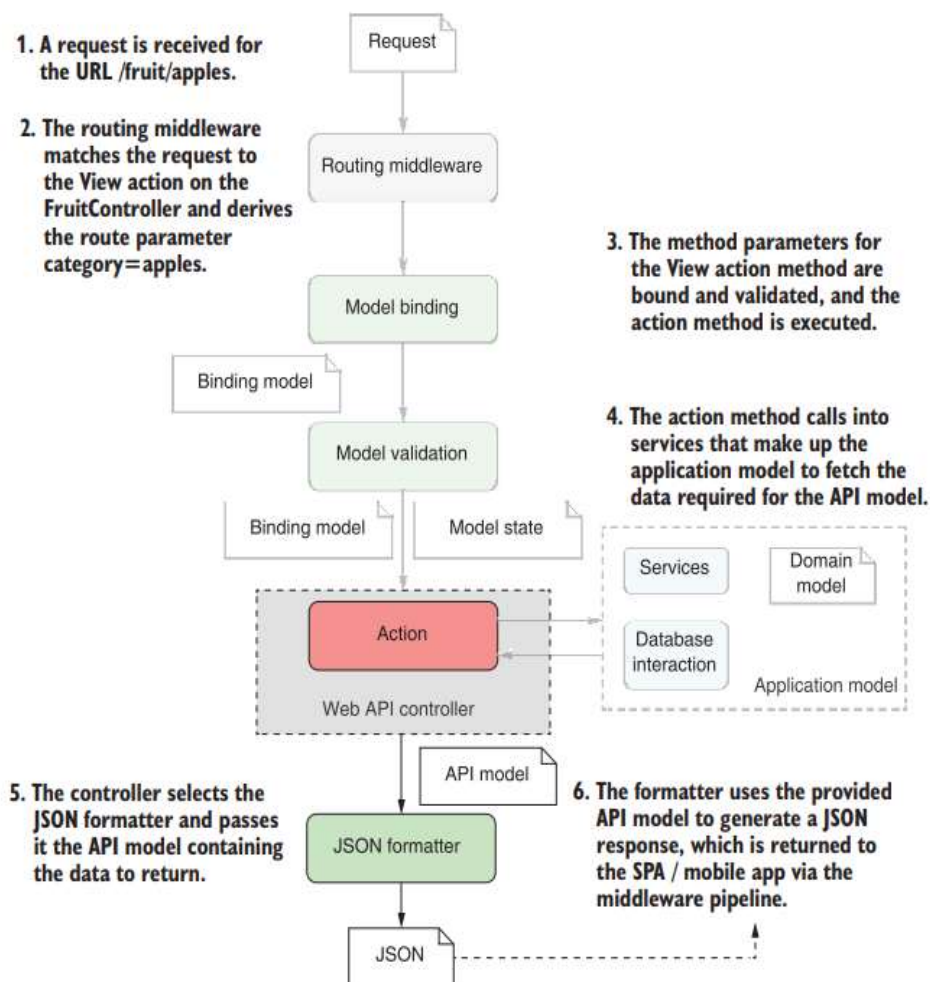


Figure 9.9 A call to a Web API endpoint in an e-commerce ASP.NET Core web application. The ghosted portion of the diagram is identical to figure 9.8.

Once the SPA is loaded in the browser, communication with a server still occurs over HTTP, but instead of sending HTML directly to the browser in response to requests, the server-side application sends data (normally in a format such as JSON) to the clientside application. The SPA then parses the data and generates the appropriate HTML to show to a user. The server-side application endpoint that the client communicates with is sometimes called a Web API.

DEFINITION A Web API exposes multiple URLs that can be used to access or change data on a server. It's typically accessed using HTTP. This is all there is to a

Web API. It exposes a number of endpoints (URLs) that client applications can send requests to and retrieve data from. These are used to power the behavior of the client apps, as well as to provide all the data the client apps need to display the correct interface to a user.

DEFINITION *View models and PageModels* contain both the data required to build a response and metadata about how to build the response. *API models* typically only contain the data to be returned in the response.

Web APIs use the same MVC design pattern, and the concepts of routing, model binding, and validation all carry through. The differentiation from traditional web applications is primarily in the view part of MVC. Instead of returning HTML, they return data as JSON or XML, which client applications use to control their behavior or update the UI. Web APIs are normally accessed from code by SPAs or mobile apps, but by accessing the URL in your web browser directly, you can view the data the API is returning. The ability to easily build a generalized HTTP Web API presents the possibility of using ASP.NET Core in a greater range of situations than can be achieved with traditional web apps alone.

ASP.NET Core 5.0 apps also include a useful endpoint for testing and exploring your Web API project in development called Swagger UI. This lets you browse the endpoints in your application, view the expected responses, and experiment by sending requests.

NOTE Swagger UI is based on the industry standard OpenAPI specification (previously called Swagger, www.openapis.org), which is enabled by default in

Web API apps. OpenAPI provides a way of documenting your API, so that you can automatically generate clients for interacting with it in dozens of different languages. For more on OpenAPI and Swagger in ASP.NET Core apps, see Microsoft's documentation: <http://mng.bz/QmjR>.

Listing 9.1 The Startup class for the default Web API project

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();
        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo {
                Title = "DefaultApiTemplate", Version = "v1" });
        });
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c => c.SwaggerEndpoint(
                "/swagger/v1/swagger.json", "DefaultApiTemplate v1"));
        }

        app.UseHttpsRedirection();
        app.UseRouting();
        app.UseAuthorization();

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

Adds services required to generate the Swagger/OpenAPI specification document

AddControllers adds the necessary services for API controllers to your application.

Adds Swagger UI middleware for exploring your Web API

MapControllers configures the API controller actions in your app as endpoints.

You can create a new Web API project in Visual Studio using the same New Project process you saw previously. Create a new ASP.NET Core application providing a project name, and, when you reach the New Project dialog box,

select the ASP.NET Core Web API template. If you're using the CLI, you can create a similar template using the command:

```
dotnet new webapi -o FirstWebAPI
```

The Startup.cs file in listing 9.1 instructs your application to find all API controllers in your application and to configure them in the endpoint Middleware. Each action method becomes an endpoint and can receive requests when the RoutingMiddleware maps an incoming URL to the action method.

Listing 9.2 A simple Web API controller

```
[ApiController]
public class FruitController : ControllerBase
{
    List<string> _fruit = new List<string>
    {
        "Pear",
        "Lemon",
        "Peach"
    };

    [HttpGet("fruit")]
    public IEnumerable<string> Index()
    {
        return _fruit;
    }
}
```

Web API controllers typically use the [ApiController] attribute to opt in to common conventions.

The ControllerBase class provides several useful functions for creating IActionResults.

The data returned would typically be fetched from the application model in a real app.

The [HttpGet] attribute defines the route template used to call the action.

The controller exposes a single action method that returns the list of fruit.

The name of the action method, Index, isn't used for routing. It can be anything you like.

Web APIs typically use the [ApiController] attribute (introduced in .NET Core 2.1) on API controllers and derive from the ControllerBase class. The base class provides several helper methods for generating results, and the [ApiController] attribute automatically applies some common conventions, as you'll see in section 9.5.

TIP There is also a Controller base class, which is typically used when you use MVC controllers with Razor views. That's not necessary for Web API controllers, so ControllerBase is the better option.

In listing 9.2, data is returned directly from the action method, but you don't have to do that. You're free to return an `ActionResult` instead, and often this is required. Depending on the desired behavior of your API, you may sometimes want to return data, and other times you may want to return a raw HTTP status code, indicating whether the request was successful. For example, if an API call is made requesting details of a product that does not exist, you might want to return a 404 Not Found status code. You're free to return any type of `ActionResult` from your Web API controllers, but you'll commonly return `StatusCodeResult` instances, which set the response to a specific status code, with or without associated data. `NotFoundResult` and `OkResult` both derive from `StatusCodeResult`, for example. Another commonly used status code is 400 Bad Request, which is normally returned when the data provided in the request fails validation. This can be generated using a `BadRequestResult`. In many cases the `[ApiController]` attribute can automatically generate 400 responses for you.

Once you've returned an `ActionResult` (or other object) from your controller, it's serialized to an appropriate response. This works in several ways, depending on

- The formatters that your app supports
- The data you return from your method
- The data formats the requesting client can handle

See the codes for further illustrations.

Summary

- A Web API exposes a number of methods or endpoints that can be used to access or change data on a server. It's typically accessed using HTTP by mobile or client-side web applications.
- Web API action methods can return data directly or can use `ActionResult<T>` to generate an arbitrary response.
- If you return more than one type of result from an action method, the method signature must return `ActionResult<T>`.
- Web APIs follow the same MVC design pattern as traditional web applications. The formatters that generate the final response form the view.
- The data returned by a Web API action is called an API model. It contains the data the middleware will serialize and send back to the client. This differs from view models and `PageModels`, which contain both data and metadata about how to generate the response.

- Web APIs are associated with route templates by applying `RouteAttributes` to your action methods. These give you complete control over the URLs that make up your application's API.
- Route attributes applied to a controller combine with attributes on action methods to form the final template. These are also combined with attributes on inherited base classes. You can use inherited attributes to reduce the amount of duplication in the attributes, such as where you're using a common prefix on your routes.
- By default, the controller and action name have no bearing on the URLs or route templates when you use attribute routing. However, you can use the "[controller]" and "[action]" tokens in your route templates to reduce repetition. They'll be replaced with the current controller and action name.
- The `[HttpPost]` and `[HttpGet]` attributes allow you to choose between actions based on the request's HTTP verb when two actions correspond to the same URL. This is a common pattern in RESTful applications.
- The `[ApiController]` attribute applies several common conventions to your controllers. Controllers decorated with the attribute will automatically bind to a request's body instead of using form values, will automatically generate a 400 Bad Request response for invalid requests, and will return `ProblemDetails` objects for status code errors. This can dramatically reduce the amount of boilerplate code you must write.
- You can control which of the conventions to apply by using the `ConfigureApiBehaviorOptions()` method and providing a configuration lambda. This is useful if you need to fit your API to an existing specification, for example.
- By default, ASP.NET Core formats the API model returned from a Web API controller as JSON. Virtually every platform can handle JSON, making your API highly interoperable.
- In contrast to the previous version of ASP.NET, JSON data is serialized using camelCase rather than PascalCase. You should consider this change if you get errors or missing values when migrating from ASP.NET to ASP.NET Core.
- ASP.NET Core 3.0 onwards uses `System.Text.Json`, which is a strict, high performance library for JSON serialization and deserialization. You can replace this serializer with the common `Newtonsoft.Json` formatter by calling `AddNewtonsoftJson()` on the return value from `services.AddControllers()`.
- Content negotiation occurs when the client specifies the type of data it can handle and the server chooses a return format based on this. It allows multiple clients to call your API and receive data in a format they can understand.
- By default, ASP.NET Core can return `text/plain`, `text/html`, and `application/json`, but you can add additional formatters if you need to support other formats.