The HTTP Cycle Tutorial

What Is HTTP?

HTTP, which stands for HyperText Transfer Protocol, is the communication protocol that powers the web. It defines how clients (such as browsers or frontend applications) and servers (such as backend APIs or web services) exchange data.

The Client-Server Model

To understand HTTP, we need to understand the client-server architecture:

- The client is the part of your app that the user interacts with, in this case, a Blazor frontend.
- The server is responsible for storing data, processing logic, and responding to client requests. In this app, this is an ASP.NET Core.

HTTP sits between them and acts as the translator and courier:

- The client sends an HTTP request asking for data or asking the server to do something.
- The server processes the request and returns an HTTP response with the result.

So why is it so vital?

HTTP is essentially acting as the language of the web, and the request-response cycle is the conversation.

Why Use It?

- All web technologies understand HTTP, from browsers to mobile apps
- It is platform independent, meaning a JavaScript frontend can talk to a .NET backend. A mobile app can talk to a Python server. HTTP unifies them all.
- Each request contains all the info the server needs, so servers don’t need to track past requests, making scaling easier.
- Status codes are easy to debug and understand.

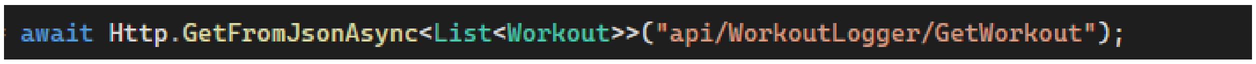
How HTTP Works

An HTTP interaction happens in two parts:

1. The Request (from the client)

It includes:

- A method (e.g., GET, POST, PUT, DELETE) to specify the action
- A URL or endpoint
- Optional headers (like content type or authentication)
- Optional body (for sending data, e.g., JSON)



```
await Http.GetFromJsonAsync<List<Workout>>("api/WorkoutLogger/GetWorkout");
```

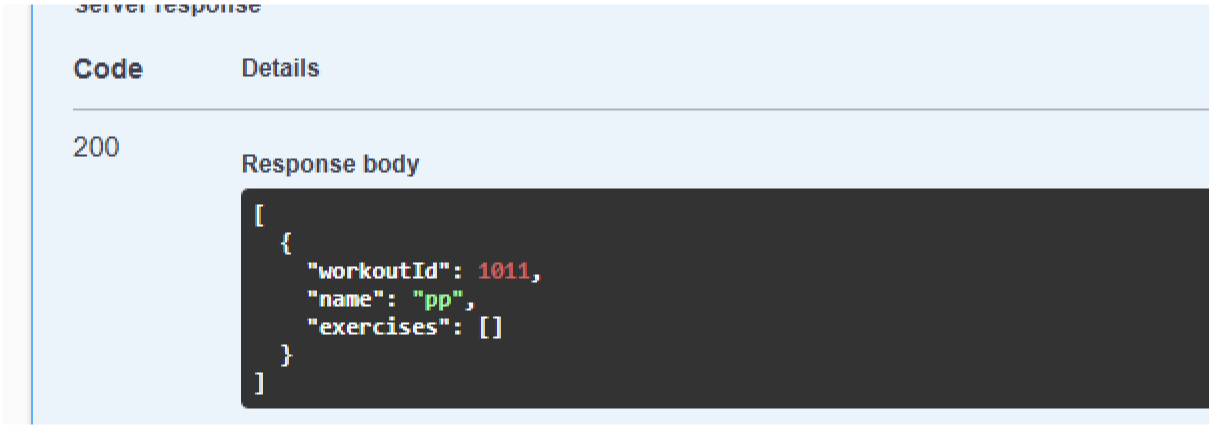
This is a GET request asking for a list of workouts.

- Constructing the Request
    - In Blazor, constructing an HTTP request is done through the HttpClient service. For read operations like retrieving data, we use the method GetFromJsonAsync<T>(), which sends an HTTP GET request and expects the server to respond with JSON that matches the type T.
        - We also have to specify the Endpoint/Url which helps map to our backend controller

2. The Response (from the server)

It includes:

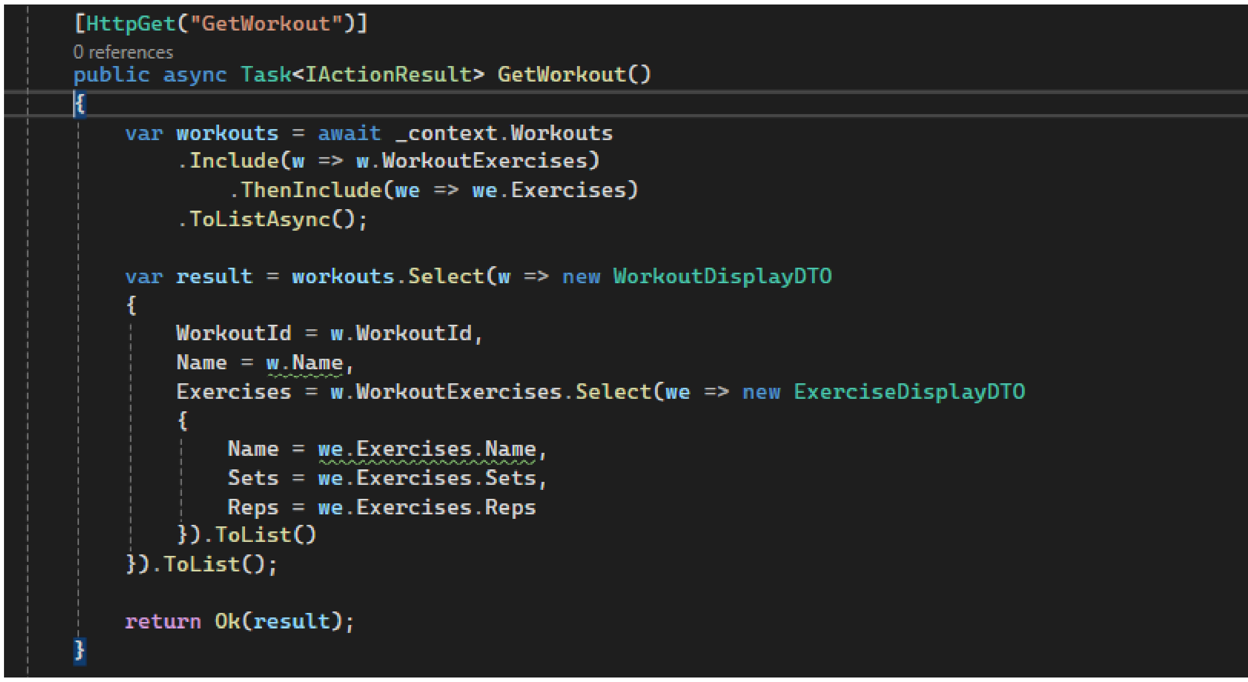
- A status code (200 OK, 404 Not Found, etc.)
- Optional headers (e.g., content type)
- Optional body — usually JSON — with the data the client requested



Server response

Code	Details
200	Response body

```
[
  {
    "workoutId": 1011,
    "name": "pp",
    "exercises": []
  }
]
```

- Constructing the Response
  - In Blazor, constructing an HTTP request is done through the HttpClient service. For read operations like retrieving data, we use the method GetFromJsonAsync<T>(), which sends an HTTP GET request and expects the server to respond with JSON that matches the type T.
  - 

```
[HttpGet("GetWorkout")]
0 references
public async Task<IActionResult> GetWorkout()
{
    var workouts = await _context.Workouts
        .Include(w => w.WorkoutExercises)
            .ThenInclude(we => we.Exercises)
        .ToListAsync();

    var result = workouts.Select(w => new WorkoutDisplayDTO
    {
        WorkoutId = w.WorkoutId,
        Name = w.Name,
        Exercises = w.WorkoutExercises.Select(we => new ExerciseDisplayDTO
        {
            Name = we.Exercises.Name,
            Sets = we.Exercises.Sets,
            Reps = we.Exercises.Reps
        }).ToList()
    }).ToList();

    return Ok(result);
}
```

HTTP is KEy to the Backend and Frontend Connection/Relationship

Without HTTP, your Blazor app would have no way of communicating with your ASP.NET Core backend. It's through this protocol that:

- You retrieve data to display (via GET)
- You send new data to save (via POST)
- You update records (via PUT)
- You remove things (via DELETE)