**Protocolo HTTP**

Ele é um marco da criação da WEB, que gira em torno da conversa do “**Client**” com o “**Server**”. Você faz um pedido (request) e ele te dá uma response(resposta).

**HTTP REQUEST**

**Verb** – São os verbos que utilizamos nas requisições (POST, GET).

**URI** – Endereço da requisição.

**Version** – Versão do protocolo.

**Request HEADER** – Cabeçalho da Requisição (versão do browser, versão do Windows etc.).

**Request Message** – É a requisição de fato.

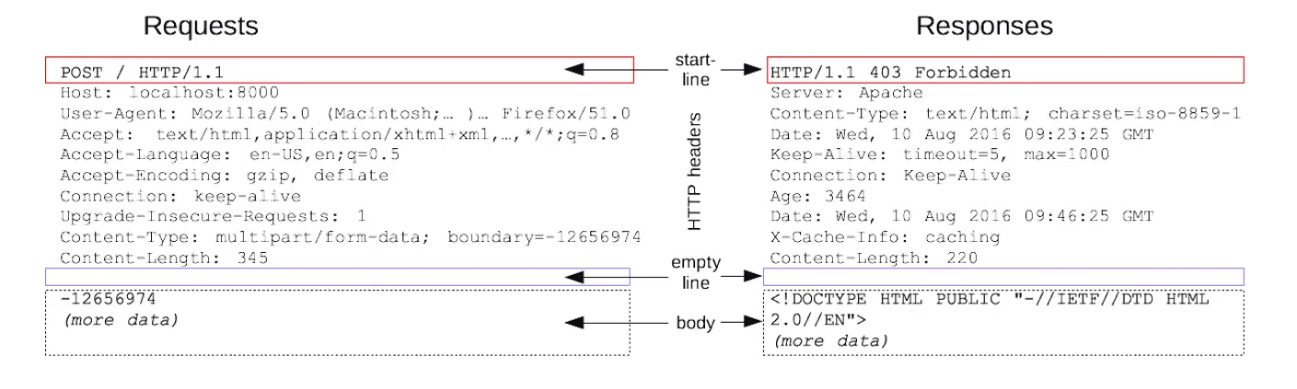
**HTTP RESPONSE**

**Response Code** - Código da resposta.

**HTTP Version** – Versão do Http.

**Response Header** – um código, um token etc.

**Response Message** – Requisição vinda do server.



**TCP – Transmission Control Protocol**

**TCP (Transmission Control Protocol)** is an important network protocol that lets two hosts connect and exchange data streams. TCP guarantees the delivery of data and packets in the same order as they were sent. Vint Cerf and Bob Kahn, who were DARPA scientists at the time, designed TCP in the 1970s.

**TLS** - **Transport Layer Security**

**Transport Layer Security (TLS)**, formerly known as **Secure Sockets Layer (SSL)**, is a protocol used by applications to communicate securely across a network, preventing tampering with and eavesdropping on email, web browsing, messaging, and other protocols. Both SSL and TLS are client / server protocols that ensure communication privacy by using cryptographic protocols to provide security over a network. When a server and client communicate using TLS, it ensures that no third party can eavesdrop or tamper with any message.

**REST vs SOAP**

**REST** – Envio de dados com base em REST (Representational State Transfer). Esse estado de representação é apenas texto (recebida e enviada). Ela é transmitida por meio do protocolo HTTP como se fossem um envelope (por possuir um head, body etc.).

**SOAP** – Simple Object Access Protocol. Além dos dados ele é um padrão baseado em XML sendo bastante verboso, tornando bem mais pesada.

**Arquitetura REST**

Uma arquitetura REST é criar um nível de abstração dentre as suas APIs. Quando falamos em arquitetura REST, não estamos dizendo como tudo funciona, mas sim que a lógica de que escrevemos está distribuída em APIs (serviços).

**Arquitetura REST em Microservices**

É uma arquitetura REST em que, cada API, possui um microservice. Um serviço que efetua apenas uma coisa. Quebrando um sistema monolito em pedaços.

**Arquitetura REST Monolítica**.

Um sistema único, não dividido, que roda em um único processo, uma aplicação de software em que diferentes componentes estão ligados a um único programa dentro de uma única plataforma.

**CRIANDO UMA API REST**

1º Criando através do dotnet CLI (command line)

1. Abro o command line or PowerShell na pasta desejada.
2. Digito o comando dotnet new -h (para obter ajuda dos demais comandos)
3. **Para criar uma WebAPI**, utilizo o comando **“dotnet new webapi”.**
4. Comando **“cd + nome da pasta”**: entra na pasta desejada.
5. Comando **“dir”** mostra todos os arquivos contidos naquele local.

**CURIOSIDADES:**

* Ao trabalhar com as APIs, elas também têm suas convenções, assim como o MVC. Com as APIs, a **Controller** recebe uma annotation **[ApiController**].
* Os métodos recebem os nomes dos verbos, já indicando a sua finalidade. Exemplo (public void Post, Get,Put, Delete).

**CONTROLLER**

A controller no desenvolvimento de APIs elas são de extrema importância, pois são com elas que temos uma base de nossa API.

A nossa **controller** herda de **ControllerBase**, parecida com uma controller padrão, porém, um pouco mais simples. Com isso, ela é complementada com a annotation **[ApiController].**

**ROTAS**

É comum em nossas annotation verbs, elas passarem uma rota, isso é muito comum em projetos. Exemplo: [HttpGet({id:int})]. É recomendado já especificarmos o tipo de dado que será aceito pela URL.

**ActionsResult**

* Quando estamos trabalhando com nossos retornos de nossas views, nos podemos utilizar o ActionResult tipado (ActionResult<IEnumarable<>) ou não (ActionResult). Porém, há uma diferença que pode influenciar, a ActionResult permite retornar alguns resultados como **BadRequest, NotFound, Ok** todos eles necessitam de um Result, ao contrário de um método IEnumerable.
* Outro motivo é que quando fazemos uma **ActionResult** e não **especificamos o tipo de retorno** dele, ele apenas receberá um **resultado**, não nos possibilitando em umas ocasiões por exemplo, de retornamos uma coleção.

**Formatadores de respostas**

* **[FromBody]** nos diz que o que estamos recebendo está vindo no corpo do nosso request.
* **[FromRoute]** podemos deixar explicito que o Id por exemplo, está vindo da rota.
* **[FromForm]** está vindo de um formulário por exemplo.
* Temos um formatador de resposta que utilizamos como annotation é o **[ProducesResponseType(typeof(nomeDaEntidade), StatusCodes.Status201Created)]**. A diferença para 201 para o 200, é que o 201 ele retorna um Ok que foi criado, já o 200 apenas uma confirmação pra qualquercoisa.
* **[ProducesResponsesType(StatusCodes.Status400BadRequest)]** para qualquer erro obtido.

**Formatadores de dados de resposta Personalizados**

* Podemos criar uma classe “mãe” que herda de nossa ControllerBase, tornando assim a nossa principal.

**Public abstract class MainController : ControllerBase**

**{**

**Protected ActionResul CustomResponse**

**}**

* Assim, podemos criar nossa ActionResult personalizada.

**[CONFERIR VIDEO FORMATADOR DE RESPONSE PERSONALIZADO PARA ENTENDER MAIS A FUNDO]**