Digital Innovation One – bootcamp GFT Start #3



**\*As anotações contidas neste doc foram feitas exclusivamente para fins de estudo pessoal**

**INTRODUÇÃO AO ECOSSITEMA .NET E DOCUMENTAÇÃO**

**Visão geral do Ecossistema .NET**

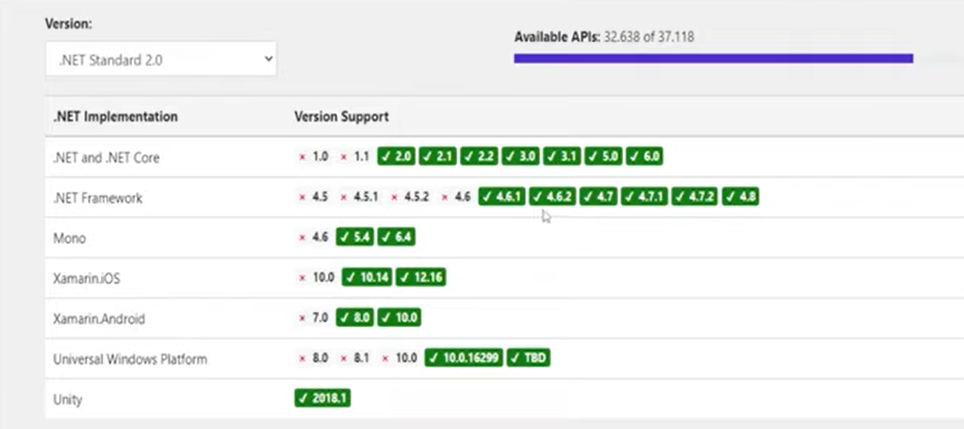
- .NET é uma plataforma de desenvolvimento com diversas ferramentas, linguagens de programação e bibliotecas que permitem o desenvolvimento de diversos tipos de aplicações

- Já o ASP.NET é um framework que estende .NET para a criação de web apps

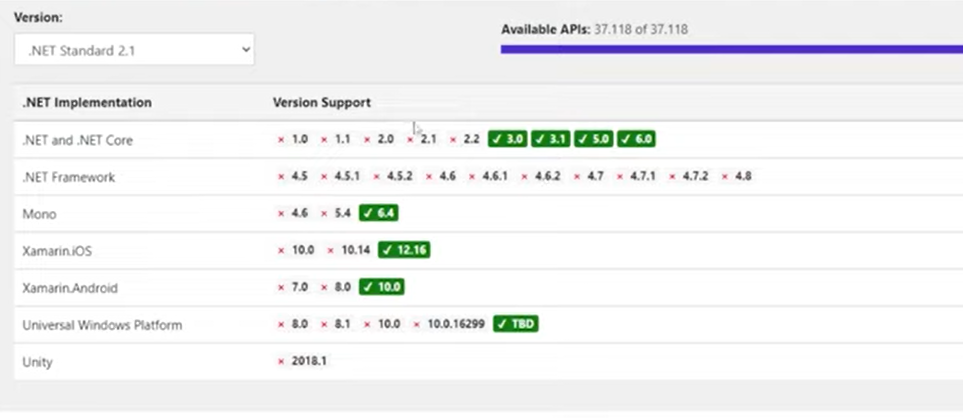
- Ou seja, o ASP.NET está dento do .NET (?)

**.NET Standart**

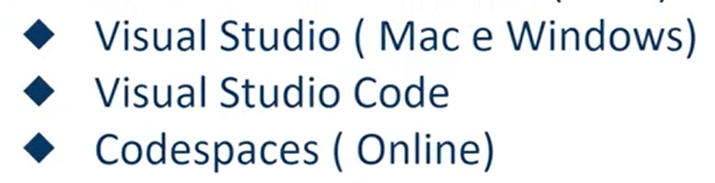
- O dotnet utiliza especificações das APIs que são comuns às implementações dotnet]



- **Exemplo:** caso uma app use a versão 2.1 do .NET não é possível conversar com .NET framework



- Ambientes de desenvolvimento:



- NUGET: gerenciador de pacotes

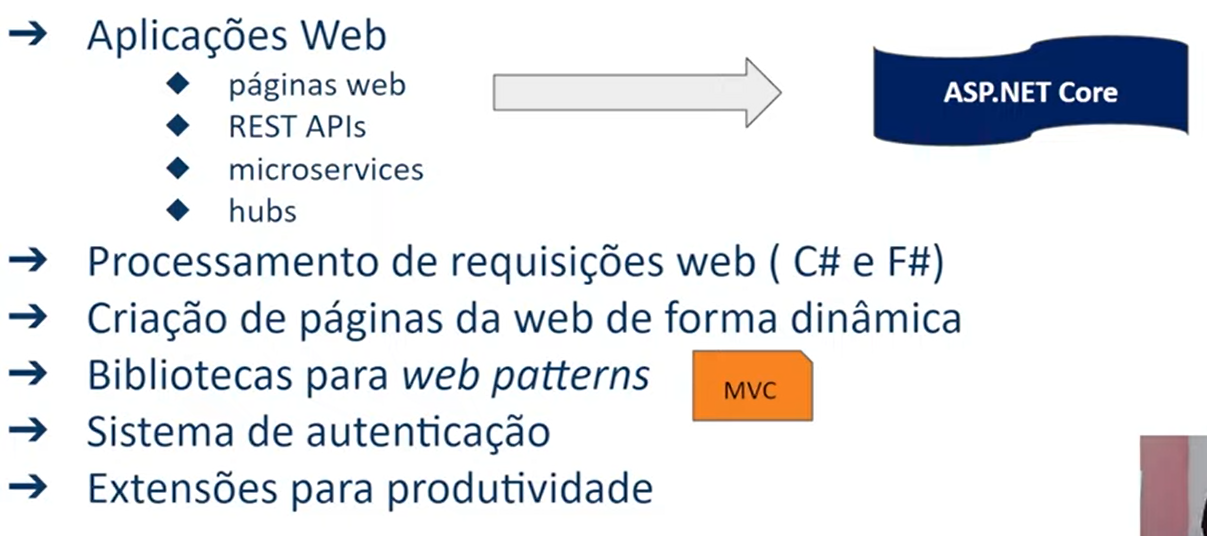
- Acesso de dados:

* ORM -> Entity Framework Core
  + Pega o código e transforma em tabela
* LINQ -> Consulta integrada à linguagem

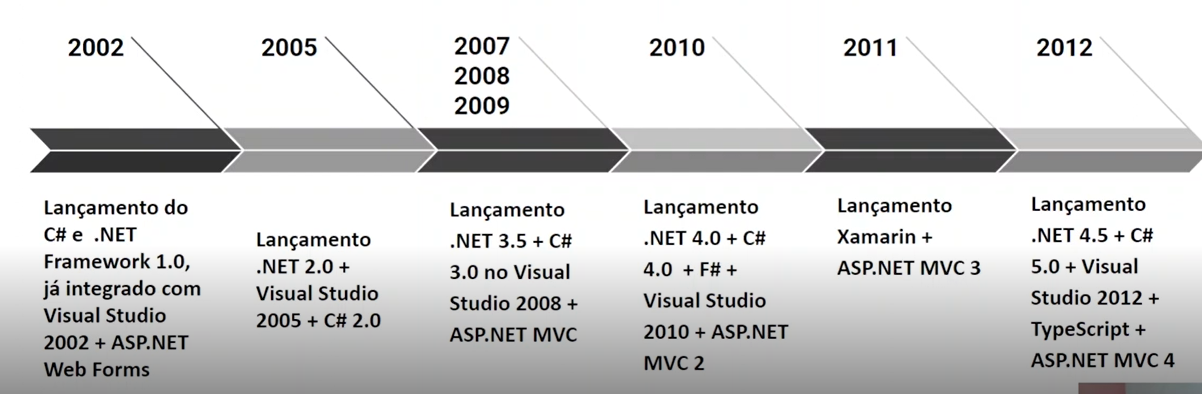
- Ambientes CI/CD

* GitHub Actions
* Azure Devops
* Cake/Fake





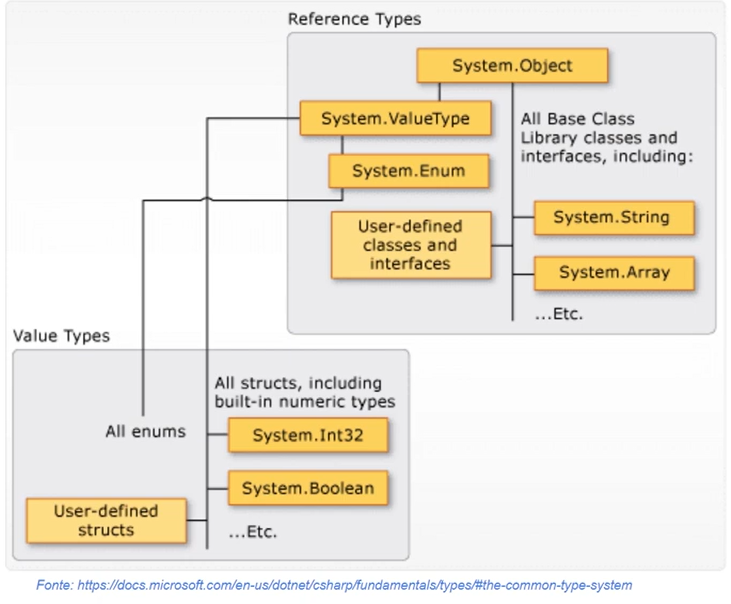
**Linha do Tempo do .NET**



-

**TRABALHANDO COM TIPOS DE REFERÊNCIA E VALOR EM C#**

- A diferença entre o que é **value type** e **reference type** o .NET chama de **Common Type System (CTS)**



- Basicamente, tudo que herda diretamente de **System.Object** é reference type

-> Já o que herda diretamente de **System.ValueType** é value type

**Definindo Value e Reference Types**

**Definindo Value Types**

- Contém uma **INSTÂNCIA** do tipo criado

- Essa instância sempre é **copiada** ao atribuir o valor para outra variável

- A alocação é na **Stack** (garante melhor performance)

- O valor inicial é sempre o **valor default** de cada tipo

**Definindo Reference Types**

- Aqui temos uma **REFERÊNCIA** para uma instância do tipo criado

- A referência **nunca muda** ao atribuir o valor para outra variável

- Na Stack fica apenas a referência para os valores (como se fosse um ponteiro) e a alocação é no **Heap**

- Seu valor inicial default é sempre “Null”

- Requer **gerenciamento da Memória** através do GC (Garbage Colector)

**Value Types**

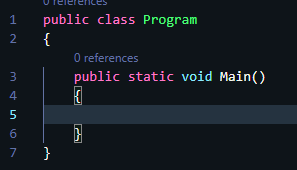
|  |  |
| --- | --- |
| Numéricos | sbyte, short, int, long, byte, ushort, uint, ulong |
| Caracteres Unicode | char |
| Pontos flutuantes | float, double, decimal |
| Booleano | bool |
| Outros… | enum, struct, nullable, tuples |

**Reference Types**

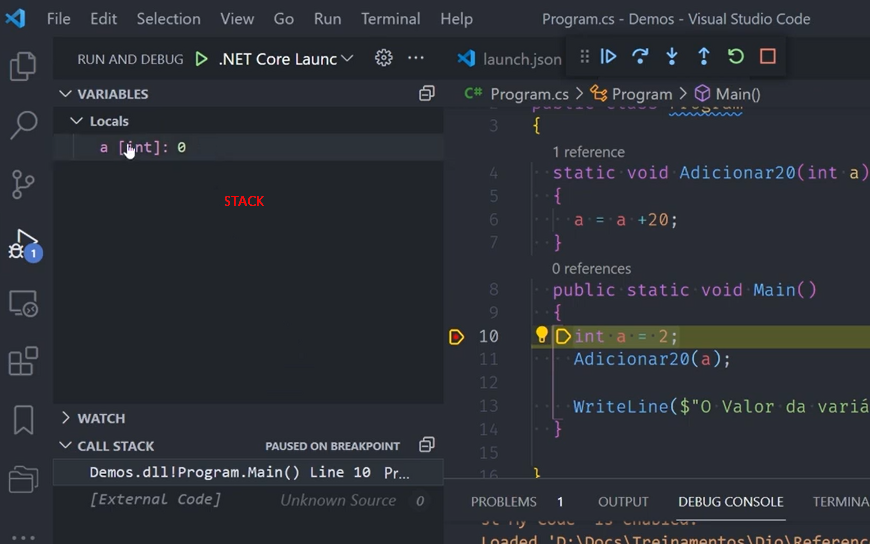
|  |  |
| --- | --- |
| Tipos Classe | class, object, string |
| Tipos Arrays | Int[], int[,], etc... |
| Outros... | Interface, delegate |

**Obs. da prática:** mudamos o “console”: “internalConsole”, no arquivo launch.json para “console”: “integratedTerminal” para que (algo?) não se misture com o log de debug;

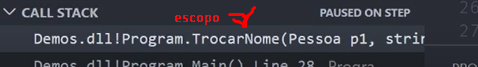
**Obs 2.:** Ctrl + L para limpar terminal (mesmo efeito que o comando cls); inicando com a classe Program no VS Code:

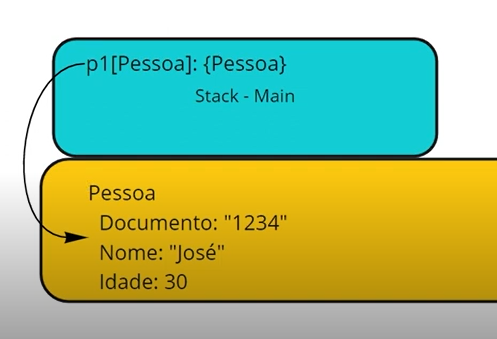


- Essa parte é a stack (ou heap em reference types)



- “Escopo Trocar Nome”





- Veja bem, quando o GC vê que tem alguma coisa ocupando memória na Heap e essa coisa não tem nenhuma referência mais no código, ele é responsável por remover para liberar a memória

- Colocando @ antes dá string no cw é possível dar enter na string que está sendo escrita e ele vai printar o texto respeitando os parágrafos que você delimitou

**Utilizando a palavra-chave “ref”**

- O “ref” indica que o conteúdo de determinada variável acessado será acessado por referência

- Pode ser usada em **4 situações**:

* Na declaração dos **parâmetros do método** e **na chamada do método**
* Na **declaração do retorno** do método
* No **corpo do método** para receber um **retorno com ref**
* Na declaração de uma **struct**

**Aprendendo ref struct**

- Um ref struct serve para garantir que a struct ficará na stack e nunca irá para heap

- Perceba que aqui a ref atua na stack, não na heap

**Limitações**

- ref struc é um objeto limitado e **não pode**:

* ser elemento tipado de um array
* ser o tipo em campo em uma classe ou não-ref struct
* implementar interfaces
* ser convertida para Object e nem para e nem para Value Type
* ser usada em métodos assíncronos

**Quando usar?**

1) Quando for necessário garantir que a instância da struct não irá acessar a heap (exemplo?)

2) Quando for usar tipos do C# que são ref struct, como o caso do ref struct Span

- Span é um ref struct muito usado quando vamos usar, por exemplo, um array e queremos garantir que ele vai ficar na stack

- Usado principalmente quando não queremos precisar usar o Garbage Collector

- Quando usamos ref struct, todos os seus descendentes e ascendentes precisam ser ref struct

**Funcionamento básico: Garbage Collector (GC)**

- É um suporte para a criação e destruição de objetos na Heap

**GC é dividido em 3 gerações**

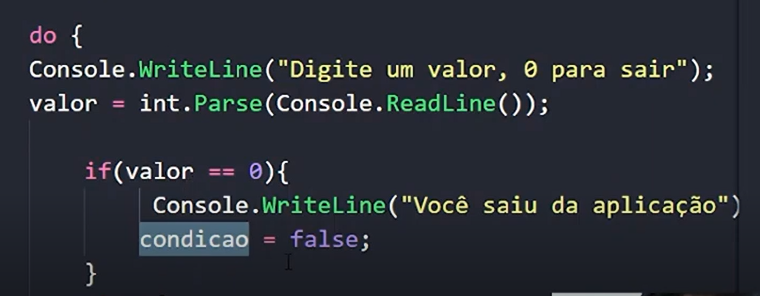
* Generation 1
  + Objetos de ciclo de vida curto
* Generation 2
  + Buffer de alternância entre Gen 1
* Generation 3
  + Objetos com longo ciclo de vida
  + Em especial, objetos criados como “static”

**CRONTROLE DE FLUXO E EXECUÇÃO**

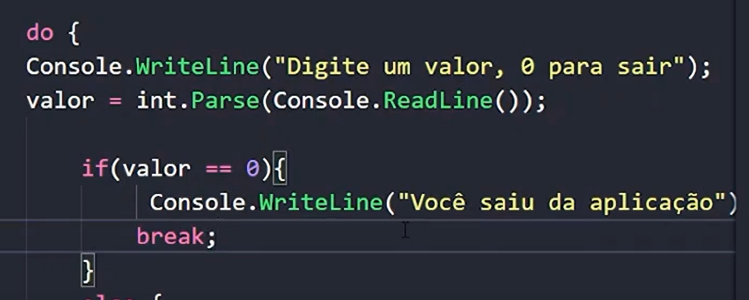
**Pulando com BREAK e CONTINUE**

- Objetivo: entender como pula uma determinada condição dentro de um bloco de laço de repetição

- Exemplo:



Vira...



- Perceba que o break, aqui, interrompe o loop e substitui a condição que foi anteriormente colocada

- Ele interrompe imediatamente a aplicação

**INTRODUÇÃO AO HTTP/HTTPS**

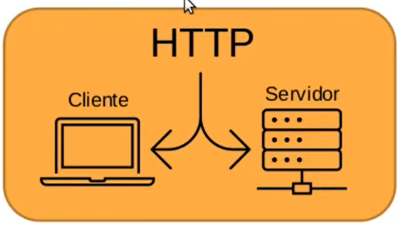
**Protocolos HTTP e SSL**

- **Objetivo:** Aprender a estrutura de funcionamento do protocolo HTTP, assim como o HTTPS

**Protocolos HTTP**

- Protocolo de comunicação

* HyperText Transfer Protocol (HTTP)
* Definido pelos RFCs 1945 e 2116
* Rege a estrutura das mensagens



- Do **lado do cliente**, o browser é que implementa o cliente HTTP

- Do **lado do servidor** (estrutural) é onde estão hospedados os objetos web (vídeos, textos, imagens...)

- O **cliente** envia mensagens do tipo **HTTP Request** para o servidor

- O **servidor** responde por mensagens do tipo **Response HTTP**

- O cliente solicita os objetos web!

- Essa comunicação ocorre através do **protocolo TCP** que é um protocolo de transporte

**O TCP opera de duas maneiras:**

**1) Conexão persistente:** uma vez estabilizada, a conexão será mantida até que haja um pedido específico de finalização da sessão

**2) Conexão não-persistente:** a cada envio de mensagem essa conexão é encerrada

- O **tipo de arquitetura** usada aqui é a Arquitetura Client-server

- O **HTTP é um protocolo Stateless**: o servidor não armazena estado do cliente

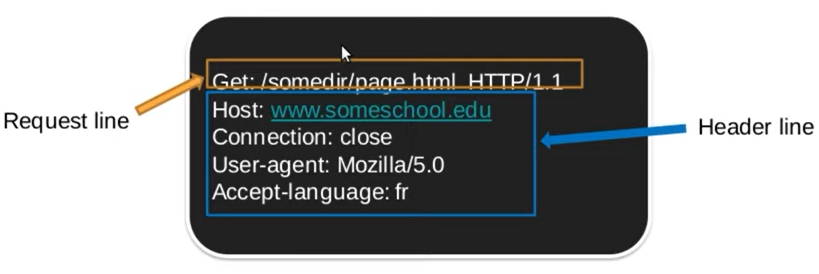
- Isso significa que ele não sabe se o cliente está enviando a mesma requisição mais de uma vez

**HTTP - Tipo Request**

- É um tipo de Texto em ASCII

- Possui dois tipos de linhas: **Request Line e Header Line**

- No **Request Line** tem o método HTTP que está sendo utilizado e a versão do protocolo



- No **Header Line**:

* a primeira linha é onde estão hospedados os dados e é o host para qual estou enviando a minha requisição
* segunda linha é a conexão (close é do tipo não persistente)
* terceira linha é o agente que realizou o request
* quarta linha é a linguagem de preferência do cliente

- Além dessas linhas, tem também o **Entity body**:

* campo da mensagem request HTTP
* não utilizada pelo GET, mas pelo método POST

- Em 90% dos casos o cliente utiliza o método GET

Métodos

|  |  |
| --- | --- |
| GET | Solicita um recurso do servidor |
| HEAD | GET sem corpo de resposta |
| POST | Submete uma entidade a um recurso |
| PUT | Substituição de recursos pelos dados da requisição |
| DELETE | Remoção de um recurso |
| TRACE\* | Chamada de loop-back a um determinado recurso |
| OPTION | Opções de comunicação com recurso |
| CONNECT | Tunelamento identificado pelo recurso |
| PATCH | Modificação parcial |

\*Através do TRACE o cliente consegue ver uma cadeia de requerimentos

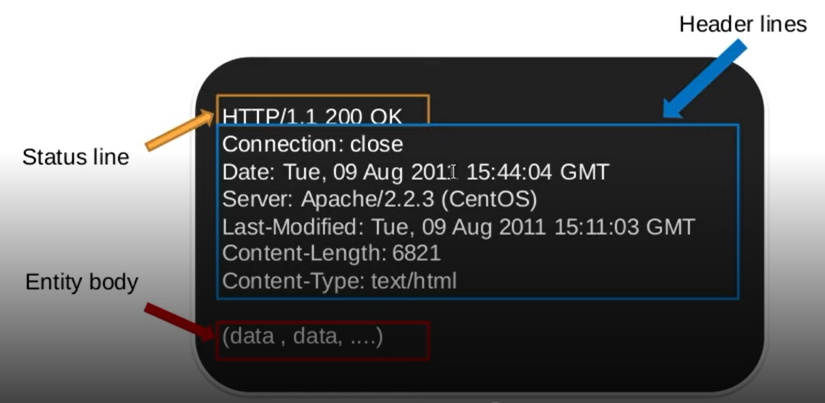
- Os **métodos seguros** são assim chamados pois não acarretam nenhuma alteração de dados no servidor



**HTTP - Tipo Response**

- Possui: Status line, header lines, entity body

- Lembrando que o Entity body não é presente no método GET



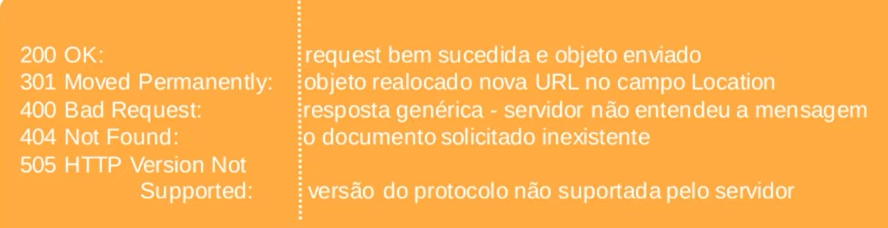
- **Status line:**

* versão do protocolo;
* status code;
* status da mensagem (OK – estamos enviando!)

- **Header line:**

* tipo de conexão;
* dados da mensagem (data, servidor...);
* content-type (tipo de dado);

**Status Code**



**Cookies**

- Apesar do HTTP ser um protocolo Stateless, existem algumas ocasiões em que é interessante reter algum tipo de identificação do usuário

- Esse **tipo de identificação** são os Cookies

- Estão referenciados na RFC 6265

- **Objetivo** de retenção dos cookies:

* Rastreamento
* Identificação dos usuários
  + Restrição ou fornecimento de funções
  + Exemplo: publicidade

**Componentes dos Cookies**

- Cookie header line:

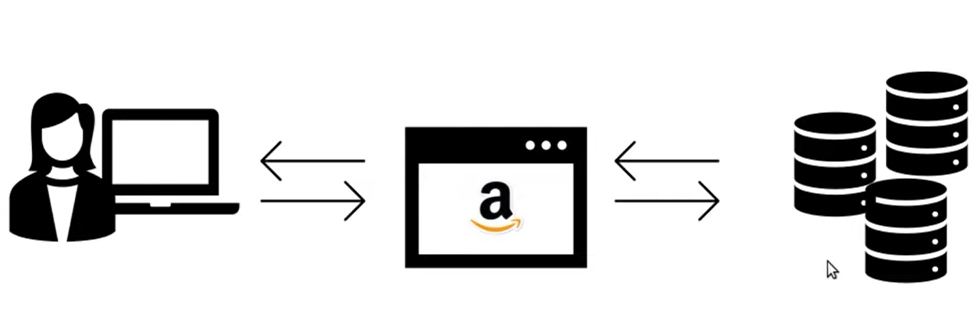
* Response message
* Request message

- Cookie file:

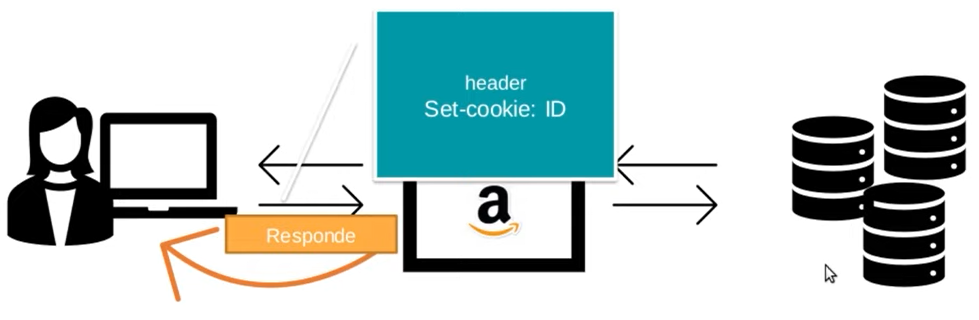
* Mantido no cliente e servidor
* No cliente, quem gerencia é o navegador

- Banco de Dados Back-end

* Do servidor
* Junto com o web-site









- A versão segura HTTPS veio para dar segurança a todo esse processo