

HTTP

Safe: não altera o estado do servidor, ou seja, é uma operação de read-only.

Idempotente: se um pedido idêntico puder ser repetido uma ou várias vezes com sequência com o mesmo efeito, deixando o server no mesmo estado.

Método	Safe	Idempotente	Cacheable
GET	✓	✓	✓
HEAD	✓	✓	✓
DELETE	✗	✓	✗
POST	✗	✗	✓
PUT	✗	✓	✗

Status Code

100 Continue

200 OK

→ sucesso

201 Created

→ criado com sucesso

300 Multiple Choice

→ mudou de sitio

301 Moved Permanently

→ encontrado no seu sitio

302 Found

→ usado em cache

304 Not Modified

→ server não entende pacote

400 Bad Request

→ user não autenticado

401 Unauthorized

→ user sem permissão

403 Forbidden

→ recurso não encontrado

404 Not Found

500 Internal Server Error

URI

schema : // [authority] path [? query] [# fragment]

↓
userinfo @ host : port

Node.js → runtime do JS; assincrono e não bloqueante

Express → framework para apps web Node.js

Express-session → session middleware usado por passport

Passport → middleware de autenticação

JS → Object-Oriented

↳ Dinâmica (verificações de tipos runtime)

↳ weakly typed

↳ Functional / Imperative / Declarative

↳ High-level ; Interpreted

```
document.querySelector('...');  
...      " "      All(...);
```

```
/:index/-search  
/:index/-doc/:doc/index
```

CSS

selector - é parte de uma CSS rule.

- padrões de elementos ou outros termos que dizer ao browser q elementos HTML serão selecionados para fazer as propriedades declaradas na CSS rule

- HTML tag
- #id
- class
- * (universal)
- tag, tag, ... (múltiplos)

Express JS

• Application → express()

• app. METHOD (path, callback)

• app. use (path, callback)

• app. listen (port, host)

• Request → req

- req. body
- req. params
- req. query
- host
- path
- method
- cookies

• Response → res

• res. cookie (name, value)

• res. clearCookie (name)

• res. status (code)

• res. send ({body}) → sets content-type inferred by body, default: text/html

• res. json ([body]) → content-type: application/json

• res. end ([data]) → não muda content-type

• res. get ('field')

• res. set ('field', 'value')

• res. redirect ([status], path)

• res. render (view, callback)

• Router → express.Router()

• Router. METHOD (path, callback)

• Router. use ([path], callback)

JS Async

• setTimeout (fun, ms) → clearTimeout (ID)

• setInterval (fun, ms) → clearInterval (ID)

• new Promise ((resolve, reject)=>{})

• Promise. resolve (data)

• Promise. reject (data)

• Promise. all (iterable)

• " . any (...) → resolve

• " . race (...) → resolve o mais rápido

Módulo JS

• eval → usado para fazer cast para number/int/string

• use strict → strict mode

this

• object method → object

• alone → global object

• function, strict → undefined

• normal → global object

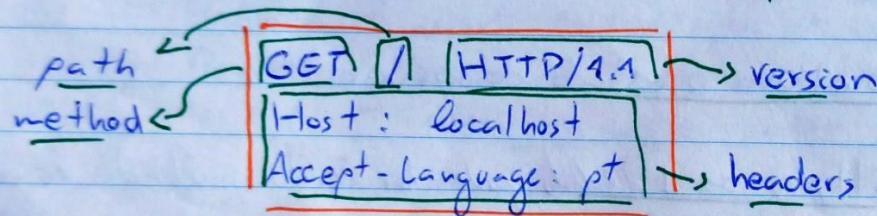
• Func (... args) → múltiplos argumentos

{HTTP} → HyperText Transfer Protocol

- ↳ usado na transferência e manipulação de recursos
- ↳ executa sobre TCP
- ↳ client-server protocol
- ↳ stateless server não mantém dados sobre pedidos
- ↳ sessionful mantém uma sessão

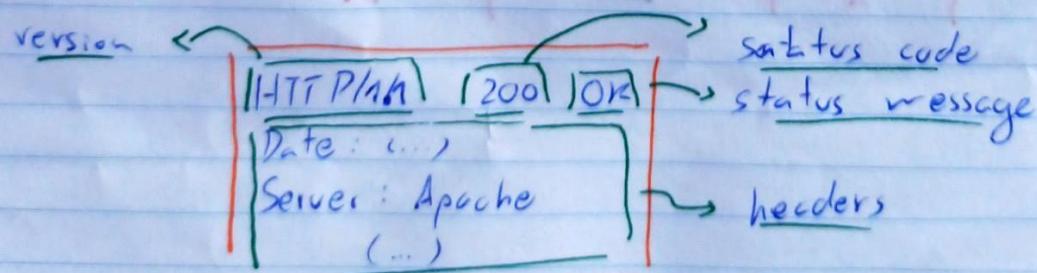
Requests

- Método - descreve a operação do cliente
- Path - path do recurso
- Version - versão do protocolo
- Headers - outra informação aos servidores (opcional)
- Body - contém o recurso enviado (apenas em alguns métodos)



Responses

- Version - versão do protocolo
- Status code - código da resposta
- Status message - descreve o status code
- Headers - informações sobre a resposta
- Body - recurso pedido (depende do método do request)



Métodos

- GET → pede a representação de um recurso
- HEAD → pede a resposta igual a um GET mas sem o body
- DELETE → apaga o recurso
- POST → envia uma entidade ao recurso, mudando o estado do server ou tendo outros side-effects
- PUT → Substitui todas as representações do recurso com o enviado
↳ também cria um novo recurso
- Propriedades:
 - Seguro (Safe) - não altera o estado do server
 - Idempotente - múltiplos ou 1 pedido pode ser feito múltiplas vezes tendo o mesmo efeito, deixando o server no mesmo estado
 - Cacheable - a resposta pode ser guardada em cache para uso futuro

Método	Safe	Idempotent	Cacheable
GET	✓	✓	✓
HEAD	✓	✓	✓
DELETE	✗	✓	✗
POST	✗	✗	✓
PUT	✗	✓	✗

Status Code

- Classes :
 - Informação → 1xx
 - Sucesso → 2xx
 - Redirecionamento → 3xx
 - Client Error → 4xx
 - Server Error → 5xx

Mais Comuns :

- 100 Continue - informa que o pedido deve continuar
- 200 Ok - Sucesso
- 201 Created - recurso criado com sucesso
- 300 Multiple Choice - pedido tem mais q uma resposta possivel
- 301 Moved Permanently - recurso mudou de sitio
- 304 Not Modified - usado em propósitos de cache
- 400 Bad Request - server não entende o pedido
- 401 Unauthorized - user not authenticated
- 403 Forbidden - user é reconhecido mas não tem permissões
- 404 Not Found - recurso não encontrado
- 500 Internal Server Error

Identificar Recursos

↳ Recursos se identificam por um URI (Uniform Resource Identifier)

↳ URI \rightarrow URL \rightarrow Uniform Resource Locator

↳ URI \rightarrow URN \rightarrow Uniform Resource Name

URLs \rightarrow mais comum; conhecido como web address;

- Scheme \rightarrow indica o protocolo o browser deve usar
- Authority \rightarrow domain name, informações do user e port

[, {
 • user info
 • host
 • port

- Path \rightarrow path do recurso
- Query \rightarrow parâmetros extra passados
- Fragment \rightarrow anchor para outra parte do recurso

| - Opcional |

Scheme :// [authority] path? [query] # [Fragment]

\Rightarrow | userinfo@host:port |

Java Script

$$\text{~} \underline{N} = -(N+1) \quad \text{Ex: } \underline{\sim 11} = -12 \quad \underline{\sim 10010} = -10011$$

~ é utilizado para casts para integer:

- $\sim\sim -1 = -1$
 - $\sim\sim \text{true} = 1$
 - $\sim\sim \text{false} = 0$
 - $\sim\sim 5.6 = 5$

Use Strict → define que o JS deve ser executado em strict mode

↳ melhora a sintaxe

↳ ajuda na escrita de um JS mais seguro

→ 'use strict'; → Apenas no inicio do script!!!

↳ Não é permitido:

- usar variáveis/objetos sem os declarar:

$$x = \underline{3.14}; \quad \text{or} \quad x = \{p_1: 10, p_2: 20\};$$

- detectar variáveis / funções:

~~do te te x ;~~

- parâmetros com o mesmo nome

- this refere-se ao objeto que invoca a função

↳ em modo normal, se o objeto não for especificado é retornado o global object (window), em strict é undefined

this → referece a new object

↳ depende de onde e como esteja ser invertido

- object method → object
 - alone → global object
 - function:
 - normal mode → global object
 - strict mode → undefined

Java Script

→ Variáveis → `var` ou `let`

Ex: `var x;`
`x = 5;`

`let x;`
`x = 5;`

`var x=5;`
`let x=5;`

`const x=5;` → Constantes

→ Tipos de dados

- String → 'hello' ; "hello" ; `hello` → Nota: {} variável ou expressão dentro da String
- Number → 3; 3,15; 3e-2
- Boolean → true ; false
(1) (0)
- null
- undefined → Variável não inicializada
- BigInt → 1n ; 900713289n → inteiro preciso
- Symbol → variável do tipo "Symbol" única e imutável
- Object → coleção de dados

Ex: `const student = {
 name: "Sérgio",
 age: 18,
 class: 10
}`

Nota: os tipos das variáveis podem mudar ao longo do programa

→ O operador typeof indica o tipo da variável

Ex: `typeof("Hello")` // retorna string

→ Output na consola

↳ função `console.log(message)`

→ Comentários

`//` - single line

`/* */` - multiple line

→ Operadores

- + - Adição
- - - Subtração
- * - Multiplicação
- / - Divisão
- % - Módulo/Resto
- ++ - Incremento
- -- - Decremento
- ** - Potência ex: $a^{**}b = a^b$

Aritméticos

- i =
- +=
- -=
- *=
- /=
- %=
- **=

de Atribuição

- ==
- !=
- ===
- !==
- >
- >=
- <
- <=

de Comparação

- && - logic AND
- || - logic OR
- ! - logic NOT

Lógicos

- delete - elimina uma propriedade em elementos de array

in

- & - AND
- | - OR
- ^ - XOR
- ~ - NOT
- << - left shift
- >> - right shift (signed)
- >>> - zero fill shift

Bitwise
Operators

→ Conversões

- numeric string + (...) = string - ex: '3' + 2 = '32'
- numeric string - , *, /, * (...) = number - ex: '3' - 1 = 2
- string - , /, * (...) = NaN
- numeric string / number + boolean = number - ex: '4' + true = 5
- number + null = number - ex: 4 + null = 4
- (...) + undefined = NaN

→ Condições

```
if(condition){  
    // code  
} else if(condition){  
    // code  
} else{  
    // code  
}
```

→ excesso de if...else

```
switch (variable(expression)){  
    case value:  
        // code  
        break; return  
    case ...  
        ...  
    default:  
        // code  
}
```

→ switch statement

→ default é opcional

Nota: se não houver break, o código após o case é executado

→ Loops

```
for (let i=1; i<=10; i++){  
    // code  
}
```

→ For loop

→ while loop

```
while (condition){  
    // code  
}
```

```
do {  
    // code  
} while (condition)
```

→ do...while loop

break: termina o loop imediatamente

continue: passa a iterar à frente (skip)

→ Funções

```
function nomeDaFuncao () {  
    // body  
}
```

parâmetros

return - retornar o valor no chamado da função

→ Objetos

```
const nome = {  
    key1: value1,  
    key2: value2  
}
```

nome[key] → value
nome["key"] → value

↳ Aceder a propriedade

↳ Definir

- Os valores podem ser variáveis, outros objetos ou funções (métodos)

→ Arrays

```
const array1 = ["eat", "sleep"]
```

→ Definir

```
const array2 = new Array("eat", "sleep")
```

• push() unshift → adicionam no fim / início repetitivamente

• pop() → remove o último; shift → remove no início

• length - retorna o size

JS Arrays

`const name = [item1, item2, ...];` → Criar array

(ou)

`const name = new Array(item1, item2, ...);`

→ Arrays são objects em que os keys são os indices

`[array].length` → retorna o tamanho do array

Métodos

- .toString()
- .join(separator) → igual ao .toString, mas pode-se especificar separator
- .push(item) → adiciona ao final
- .pop() → remove ao final
- .shift() → remove ao inicio
- .unshift(item) → adiciona no inicio
- .concat(array) → merge de arrays
- .sort(compare function) → funciona com strings by default
- .reverse() → inverte o array

Iterar

- .forEach((value, index, array) => {})
- .map(...)
- .filter(...)
- .reduce((total, value, ...) => {}, initialValue);
- .reduceRight(...)
- .every(...)
- .some(...)
- .find(...)
- .includes(item)

JS Objects

`const person = { name: "André", age: 19 };` → Criar objeto

↳ Acessar propriedades: `person.name` (ou `person["name"]`)

↳ Iterar: `for (let key in obj)` → itera pelos keys

- `Object.keys(obj)` → array de keys
- `Object.entries(obj)` → array de entries `[[k, v], [k, v], ...]`
- `Object.values(obj)` → array de values
- `Object.fromEntries(Array de entries)` → retorna objeto

Object Constructors

↳ blueprint para criar objetos do mesmo tipo

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
```

→ Criar constructor

`const me = new Person("André", 19);` → Criar instâncias

`me.nationality = "Portuguese"` → Adicionar propriedades a instâncias

↳ Não é possível adicionar métodos e propriedades aos constructos ao longo do programa, só na definir em a instâncias
da mesma maneira que nas instâncias ...

Prototype → propriedade q permite a adição de novas propriedades ou métodos ao constructor

`Person.prototype.nationality = "English";`

JavaScript Async

- ↳ Funções assíncronas são aquelas que executam em paralelo com outras funções
- ↳ setTimeout() → executa o callback quando o timer expirar
 - ↳ é passado um callback e o tempo em ms
 - ↳ é retornado um timer ID
 - ↳ setTimeout(myFun, 3000);
 - ↳ executa myFun após 3 segundos
- ↳ setInterval() → executa o callback a cada intervalo
 - ↳ é passado um callback e o tempo em ms
 - ↳ é retornado interval ID
 - ↳ setInterval(myFun, 3000);
 - ↳ executa myFun de 3 em 3 segundos
- ↳ clearTimeout() e clearInterval() recebem os IDs retornados pelas funções acima e cancelam as mesmas.

Promises

- ↳ Representa o succeso ou falhando de uma operação assíncrona, e o seu valor resultante
- ↳ Tem duas propriedades: state e result

<u>state</u>	<u>result</u>
<u>pending</u>	<u>undefined</u>
<u>fulfilled</u>	<u>result value</u>
<u>rejected</u>	<u>error object</u>

↳ Criar uma promise:

```
const p = new Promise ( function(resolve, reject) {  
    resolve ("done");  
    // ou reject (new Error ("Whoops"));  
});
```

- recebe uma função com 2 callbacks, o resolve e o reject que terminam a promise

resolve (value) → promise fulfilled
reject (error) → promise rejected

↳ Métodos

- then → executado quando a promise é resolvida

```
p.then (  
    function(result){}, // executado quando está resolvida  
    function(error){} // " " " " rejeitada  
,
```

↳ Pode ser só passado 1 argumento (o result):

- catch → usado para erros

↳ normalmente usado a seguir ao then (com 1 argumento)

```
p.then (...)  
    .catch (...)
```

↳ Analógico a then (null, function):

- finally → é executado sempre quando a promise é resolvida ou rejeitada

```
p. then (...)  
    .catch (...)  
    .finally (...)
```

Async / Await

- async → faz com q a função retorne uma promise
- await → faz com q a função espere pela promise
- ↳ Se pode ser usado nova async function

Outros Métodos

- Promise.all (iterable) → espera q todos sejam resolvidos ou q alguma seja rejeitada
- Promise.allSettled (iterable) → espera q todas estejam resolvidas ou rejeitadas
- Promise.any (iterable) → espera q alguma seja resolvida
- Promise.race (iterable) → espera q alguma seja resolvida ou rejeitada
- Promise.resolve (value) → retorna uma promise resolvida
- Promise.reject (error) → retorna uma promise rejeitada

Node.js

- ↳ back-end JS runtime environment +
- ↳ assíncrono e não bloqueante
- ↳ single threaded

node file.js → executar ficheiro

Modules → conjunto de funções que incluímos na app

require('name') → incluir módulo

module.exports = (...) → exportar um módulo

Módulos usados:

- fs
- node-fetch
- hbs
- crypto
- passport
- express
- jest

NPM → Node Package Manager

- npm install package → instalar package
- npm init → → criar projeto node

Express JS

- ↳ back-end web application framework para Node.js
- ↳ usado no desenvolvimento de websites e web APIs
- ↳ baseado em middlewares

express() → cria uma aplicação Express

```
const express = require('express');
const app = express();
```

↳ A aplicação tem métodos para:

- rotear HTTP requests
- configurar middleware
- renderizar HTML views
- registrar a template engine

Métodos

- express.json() → middleware que parses incoming requests with JSON payloads
- express.static(root) → middleware que serve static files
- express.Router() → cria um objeto Router
- express.urlencoded() → parses incoming requests with urlencoded payloads

Application Methods

- app.all(path, callback) → executado em qq método
- app.delete(path, callback) → DELETE request
- app.get(path, callback) → GET request
- app.listen(path) → inicia uma UNIX socket e espera por conexões
- app.post(path, callback) → POST request
- app.put(path, callback) → PUT request
- app.set(name, value) → atribui o valor value à propriedade name
- app.use([path], callback) → atribui o middleware

* Se for usado "*" no path, é para todos os paths

Request → representa o HTTP request

- ↳ Propriedades:
- req. body → body do request
 - req. params → parâmetros do path
 - req. path → path
 - req. query → parâmetros da query
 - req. protocol
 - req. method
 - req. host
 - req. cookies

Resposta → representa uma HTTP response

- ↳ Métodos:
- res. cookie(name, value) → adiciona um cookie
 - res. clearCookie(name) → clears the cookie
 - res. end([data]) → acaba a resposta rapidamente
 - res. get(field) → obter propriedades / headers
 - res. json([body]) → envia uma resposta JSON
 - res. redirect([status], path) → redireciona
 - res. render(view, callback) → renderiza uma view
 - res. send([body]) → envia a resposta
 - res. status(code) → muda o status code da resposta

Router → instância de middleware e rotas

- ↳ Métodos:
- router. all(path, callback)
 - router. METHOD(path, callback)
 - router. use([path], function)

Middleware → funções com acesso ao req e à res e à proxima middleware function (next)

↳ realiza as seguintes operações:

- executa seu código
- faz mudanças no request e na response
- acaba com o ciclo de middlewares
- chama a próxima middleware function

alguns split ways: using an "*" character

HTML → HyperText Markup Language

- ↳ define a estrutura do conteúdo Web
- ↳ Consiste em elementos identificados por tags
- ↳ "diz" ao browser como mostrar o conteúdo

Estrutura básica

```
<!DOCTYPE html> → tipo de documento (neste caso HTML 5)  
<html> → root element of a HTML page  
<head> → contém meta information sobre a página;  
        (...)    não visível  
</head>  
<body> → contém o corpo do documento com todos os  
        (...)    conteúdos visíveis  
</body>  
</html>
```

HTML Elements

- ↳ Constituído por start tag, content and end tag

| <tagname> content </tagname> |

⚠ Alguns elementos não têm content nem end tag, são
chamados de Empty Elements

- ↳ Os elementos podem ter atributos que adicionam informações extra
 - ↳ São especificados na start tag
 - ↳ São pares nome/valor: name = "value"

Tags mais comuns

- <!-- Comment -->
- Text → hyperlink
-
 → line break
- <button type="submit"> Text </button> → botão
- <div> → seção da página
- <h1> a <h6> → headers
- <hr> → mudança de conteúdo; normalmente cima tinha separadora
- Content </label> → label para o <input>
- <nav> (...) </nav> → navigation links
- Coffee
 - Tea → ordered list
-
- <p> Content </p> → parágrafo
- Content → section da página
- text → texto importante (bold)
- <table>
 - <tr>
 - <th> Name </th> → table header
 - <td> Ande? <td> → table data
- </tr>
- <tr>
- <td> Ande? <td> → table data
- </td>
- </table>
- <title> Title </title> → título da página (head)
- (...) → lista não ordenada
- <form action="route" method="GET"> Content </form> → formulário
- <select name="...">
 - <option value="..."> Text </option> → option
- </select>

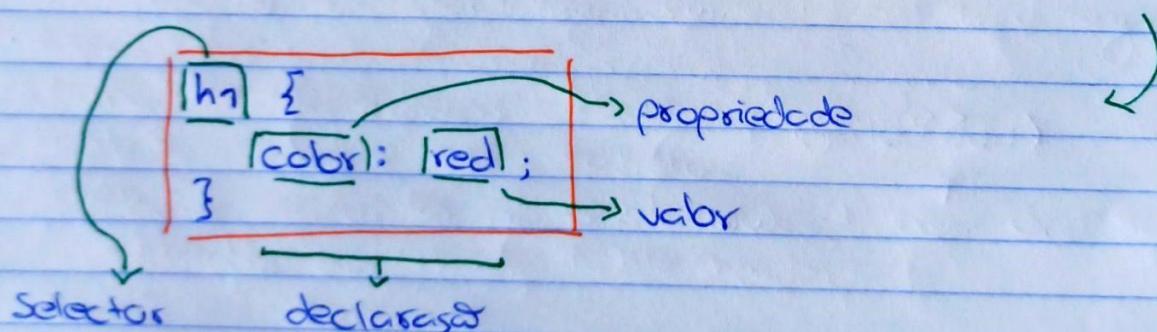
{CSS} → Cascading Style Sheets

- ↳ define a apresentação do documento HTML
- ↳ descreve como os elementos HTML são renderizados no ecrã, papel ou outros meios

Sintaxe

- ↳ Uma css rule consiste no seletor e suas declarações
- ↳ seletor - define qual (queis) elemento(s) HTML vai ser styled
 - element selector → tag name `p { ... }`
 - id selector → `#id { ... }`
 - class selector → `.class { ... }`
 - universal selector → todos os elementos `* { ... }`
 - grouping selector → múltiplos elementos `h1, h2, p { ... }`

- ↳ declarações → contém uma ou mais declarações separadas por " ; "
- ↳ cada declaração contém uma propriedade e um valor



Usar

- ↳ inline → usar o atributo style nos elementos HTML
- ↳ internal → usar a tag <style> na seção <head>
- ↳ external → usar a tag <link> na <head> e usar css externo

{Bootstrap} → CSS framework

- ↳ tem como propósito simplificar o desenvolvimento de páginas web, em front-end.

HTML DOM & Client JS

- ↳ Quando o HTML é carregado no browser, torna-se um document object
- ↳ document object é uma propriedade do window object
 - ↳ accedido com window.document ou com document

Métodos:

- getElementById('id')
- getElementsByClassName('class')
- getElementsByTagName('tag')
- querySelector('selector')
- querySelectorAll('selector')

Elastic Search

• GET	http://localhost:9200/_cat/health	→ DB health
• DELETE	/_all	→ apagar todos os índices
• GET	/_index/_search	→ pesquisar índices
• GET	/_cat/_indices	→ pesquisar índices
• DELETE	/_index	→ apagar índice
• DELETE	/_index/_doc/_doc_in_index	→ apagar doc
• GET	/_index/_doc/_doc_in_index	→ pesquisar doc
• POST	/_index/_doc/_doc_in_index	→ atualizar doc

Fetch API

- ↳ permite acceder e manipular requests e responses HTTP;
- ↳ provides the global fetch() method

Fetch (resource, [options])

- ↳ retorna promise com o response

```
{  
  method: "...",  
  headers: { ... },  
  body: ...;  
  (...)
```

File System

```
const fs = require('fs');
```

- ↳ Permite:
 - ler files → fs.readFile()
 - criar files → fs.writeFile(path, content) → escrever dado
 - update files → fs.appendFile(path, content) → adicionar no final
 - deletar files → fs.unlink(path)
 - renomear files → fs.rename('path', nome)

Sast Module → js testing framework

Nomear Gênero `(...).test.js`

- `describe('name', () => {});` → agrupa testes relacionados
- `/test('name', () => {});` → cria um teste

- `afterEach(func);` → executa ~~antes~~^{após} de cada teste
- `beforeEach(func);` → executa antes de cada teste
- `afterAll(func);` → executa ~~antes~~^{após} de todos os testes
- `beforeAll(func);` → executa antes de todo o teste

Verificar valores com expect:

- `expect(value).toEqual(expected)`
 - `.rejects.toEqual(error)`
 - `.toBeDefined()`
 - `.toBeNull()`
 - `.toBeTruthy()`
 - `.toBeFalsy()`

Handlebars

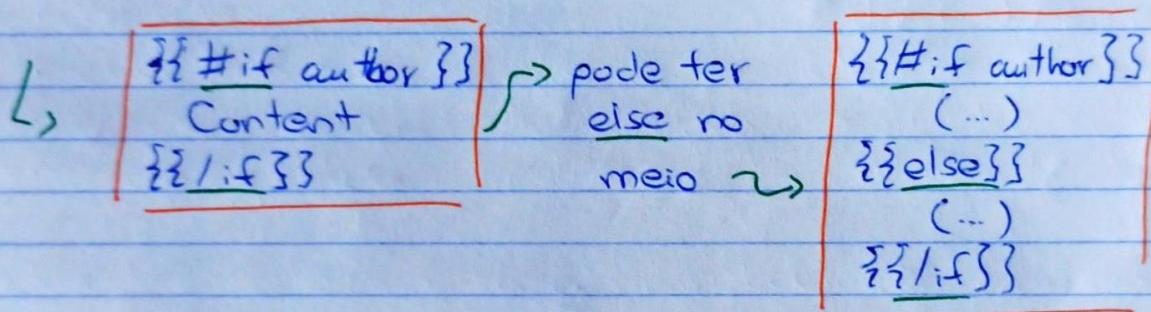
- ↳ templating language
- ↳ usa um template e um input para gerar texto HTML

Uma expressão Hbs tem a seguinte estrutura:

| {{ content }} | ↵

Helpers

- #if → renderiza um bloco condicionalmente



- unless → inverso do if; renderiza sempre exceto quando a condição é verdadeira

- #each → itera sobre uma lista

↳ this refere-se ao elemento

↳ @index refere-se ao índice

↳ @key chave do objeto

↳ @first e @last primeiro e último elemento

- #with → como o if, mas muda o this

↳ pode ter else também

⚠ Para mudar o contexto pode ser usado .. / ex: `{{{../author}}}`