



INSTITUTO FEDERAL
Sul-rio-grandense

Câmpus
Pelotas



DESENVOLVIMENTO FRONT-END II

Ferramentas de desenvolvimento Javascript (ES5/6)



Desenvolvimento Front-End II

Tópicos



- **Apresentação da Disciplina:**
 - Ementa e Plano de Ensino
 - Cronograma e Avaliações
- **Revisão Javascript**
 - Principais Conceitos
 - Versões ES5/ES6
 - Variáveis e tipos de dados
 - Functions e Arrow functions
 - Spread e desestruturação
 - Importação/Exportação de módulos
 - Objetos e Iterações de objetos
 - Funções para *Arrays*
- **Ferramentas de Desenvolvimento Front-End**
 - Ambiente de desenvolvimento: **Node**, **Npm** e **Yarn**
 - *Bundlers* e um primeiro projeto com **Vite**
- **Atividades Práticas**





- **Ementa:** *link*

A ementa é a definição da disciplina, nela encontramos o seu tema principal, assim como o programa de conteúdos de referências bibliográficas. ***Link***

- **Plano de Ensino:** *link*

Acesse o plano de ensino para saber como as definições da disciplina e o programa serão trabalhados pedagogicamente ao longo do semestre.

Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ECMAScript (JavaScript):**
 - Mas O QUE É **ECMAScript** JAVASCRIPT:
 - Linguagem interpretada criada para rodar no navegador web.
 - Não é JAVA, o nome semelhante é apenas uma questão de marketing.
 - Criada para o Netscape 2.0 por Bredan Eich
 - Evoluiu de acordo com a evolução da Web, a cada nova versão novos recursos são adicionados ou padrões anteriores alterados.
 - Padronizada pela ECMA International pela especificação ECMA-262 e ISO/IEC 16262.
 - Já esta na sua versão ES2023, mas as principais modificações foram feitas na versão ES6.



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ECMAScript (JavaScript):**

- **Javascript**® → Sun Microsystems (Oracle): marketing com lançamento do Java.
- **ECMAScript:** NetScape → ECMA262 Versão 1 – 1997.

(ECMA: *European Computer Manufactures Association*)

- **Versões:**

- 1998 – ECMAScript 2
- 1999 – ECMAScript 3
- 2008 – ECMAScript 4 (abandonada)
- **2009 - ECMAScript 5**
- **2011 - ECMAScript 5.1**
- **2015 – ECMAScript 6 (ES6)**
- 2016 - ECMAScript 7 (ES7)

- **Versões:**

- 2017 - ECMAScript 8 (ES8)
- 2018 - ECMAScript 9 (ES9)
- 2019 - ECMAScript 10 (ES10)
- 2020 - ECMAScript 10 (ES11)
- 2021 - ECMAScript 10 (ES12)
- 2022 - ECMAScript 10 (ES13)
- **2023 - ECMAScript 14 (ES14)**
- **E continua com as TCs**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **COMO EXECUTÁVAMOS PROGRAMAS JS:**

Historicamente executado (*interpretado*) em navegadores através da **tag** `<script>` do documento HTML;

Embarcado no documento HTML dentro das tags script:

```
<script> var mensagem = 'Olá Mundo!'; alert(mensagem) </script>
```

Através de um arquivo externo com a extensão .js.

```
<script src = 'caminho/do/arquivo.js'> </script>
```

Em um interpretador como o **nodejs** através do terminal ou prompt no windows.

```
>node nomedoarquivo.js
```



Desenvolvimento Front-End II

Ferramentas de desenvolvimento Front-End



- **Node e NPM:**

Durante a disciplina será necessário o uso de ferramentas de desenvolvimento, portanto precisamos do ambiente de execução *Javascript* **NodeJS** (**LTS**) e o gerenciador de pacotes **NPM**.

- Gerenciadores de versão do node possuem a vantagem de alternar facilmente entre versões rapidamente:
 - <https://nodejs.org/en/download/package-manager/current>
- **FNM** para Windows (**F**ast **N**ode **M**anager)
- **FNM** ou **NVM** para **Linux** (**N**ode **V**ersion **M**anager)
- Instalar sempre a **ÚLTIMA VERSÃO LTS (Long Term Support)**

Desenvolvimento Front-End II

Ferramentas de desenvolvimento Front-End



- **Yarn:**

O *Yarn* é outro gerenciador de pacotes, alternativo ao **NPM**. Utiliza comandos menos “verbosos”. Não é recomendado utilizar em um projeto que já foi configurado com o *npm*. Ou seja, utiliza-se um ou o outro.

- ➡ **Link e instruções de instalação**
- ➡ **Yarn vs NPM**
- *Mas para algo ainda menos verboso, experimento “ni”*
 - ➡ **documentação do “ni”**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **TIPOS DE DADOS EXEMPLOS:**

- Number: integers, float, etc;
- String: vetor de caracteres, definidos por "" ou " ";
- Boolean: true ou false;
- Null: sem valor;
- Undefined: Uma variável declarada mas sem valor definido;
- Symbol (ES6): Valor único que não pode ser igual a nenhum outro valor;
- Qualquer outro tipo é um Object;

- **EXEMPLOS:**

- <https://repl.it/@g1ll/tiposemjs>
- Um pouco mais sobre tipos primitivos em js
- Primitivos: <https://developer.mozilla.org/pt-BR/docs/Glossario/Primitivo>
- Novos tipos (ES5, ES6, ES7-8 Next): <http://modernjs.com/data-types.html>



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **DECLARANDO VARIÁVEIS:**
- Para a declaração de variáveis o JavaScript tradicional, antes do ES6, usava apenas a palavra reservada **var**. Com a atualização para o **ES 6** foram definidas as palavras chaves **let** e **const**, vejamos as diferenças:
 - **var**: usado até a especificação ES5, permite redeclarar variáveis, pode ser local ou global e independe de escopo de { }, depende apenas de escopo de função.
 - **let**: grande diferença para o var é que seu escopo é dependente de bloco, qualquer bloco { }, como por exemplo dentro de um IF.
 - **const**: escopo por bloco, assim como o let, porém não deixa redefinir a variável, ou seja, atribuir novos valores;
- Mas é possível alterar elementos internos de um Array (vetor) ou Atributos de um Objeto;
- Na maioria dos casos as variáveis serão declaradas com **let** (90%);
- Links Úteis:
 - ➡ **VAR, LET e CONST**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **DECLARANDO VARIÁVEIS - EXEMPLOS:**

- Este código funciona sem erros para **VAR**:

```
var pi = 3.14
```

```
var pi = 3 //Redeclaração é aceita sem erros
```

- Com **LET**:

```
let pi = 3.14
```

```
var pi = 3 //Erro {Identifier 'pi' has already been declared}
```

```
let pi = 0.14 //Erro {Identifier 'pi' has already been declared}
```

```
console.log(pi)
```

Mais exemplos no Link: <https://stackblitz.com/edit/g11l-js-let?file=index.js>

- Com **const**: Mais Exemplos : [stackblitz/g11l-jsconst](https://stackblitz.com/edit/g11l-jsconst)

```
const pi = 3.14
```

```
pi = 3.14 //Erro {Assignment to constant variable.}
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- TIPO NUMBER e BIGINT - EXEMPLOS:

- Para representar números o JS usa o tipo **NUMBER** que possui a precisão de um **double** de 64 bits de acordo com a especificação IEEE754 (-2^{53} a 2^{53}):

- Algumas funções úteis:

- **toFixed(numeroDeCasasDecimais)**: retorna o número formatado com o número de casas decimais passado como parâmetro. O retorno é uma String.

```
var pi = 3.1415
```

```
console.log(pi.toFixed(2)) //Imprime 3.14
```

- Maior inteiro possível (Propriedades estáticas):

- **Number.MAX_SAFE_INTEGER**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- TIPO NUMBER e BIGINT - EXEMPLOS:
- O tipo **BigInt** representa valores maiores que 2^{53}
- Possui suporte em Chrome e Firefox
 - Proposta do **BigInt**: <https://github.com/tc39/proposal-bigint>
 - Exemplos:

```
let aBigInt = 11n
```

```
let aNumber = 156
```

```
aBigInt = BigInt(aNumber) // converte para BigInt
```

```
aBigInt === 156n // true
```

```
typeof 156
```

```
// "number"
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- TIPO NUMBER e BIGINT - EXEMPLOS:
- O tipo **BigInt** representa valores maiores que 2^{53}
- Ainda não é recomendado usar se a aplicação não trabalha com valores maiores do que o limite do **Number** (REF).
- Mais exemplos no Link:
 - *código do exemplo*

```
let aBigInt = 11n
let aNumber = 156
aBigInt = BigInt(aNumber) // converte para
BigInt
aBigInt === 156n // true
typeof 156
// "number"
typeof 156n
// "bigint"
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- TIPO STRING - EXEMPLOS:
- O objeto global **String** é um construtor para **strings**, ou uma sequência de caracteres:

```
- const texo1 = "olá" //Criação literal com “”  
  const texto2 = 'mundo' //Criação literal com ‘’  
  const mensagem = String(texto1 + texto2) //Objeto Global
```

- **Concatenação**

```
- console.log( '1' + 1) //imprime 11 no console  
- console.log(texto1 + texto2)
```

- **Templates literals :**

```
console.log(`0 programa diz: ${mensagem}`)
```

- `console.log(`0 valor de PI é ${Math.PI.toFixed(2)}`)`

- Mais Exemplos : `g11l-js-strings`



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **FUNCTIONS E ARROW FUNCTIONS:**

- Mudança na sintaxe

- **FUNCTION:**

- `function nomeDaFuncao(param){return param}`

- `function (param){return param} //anônima`

- Ex: `function quadrado(n){return n**2}`

- **ARROW:**

- `const | let nomeDaFuncao = (param) => {return param}`

- `nomeDaFuncao = param => expressao+de+retorno`

- Ex: `const quadrado = n => n**2`

- **Function** possui sua própria referência **THIS**, uma função também é um objeto.

- **Arrow** functions **NÃO TEM THIS**, recebendo o **this** do escopo em que ela foi declarada.

Exemplos: g111-js-fnc-arrow



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ARRAYS:**

- Array é um Objeto em JS com a implementação de uma Lista de pares índice/valor ou chave/valor:

```
const carros = ["gol", "palio", "celta"]  
let frutas = Array("Laranja", "Maçã", "Uva")  
let nuns = Array("Laranja", "Maçã", "Uva")
```

- O objeto **Array** possui métodos que facilitam a manipulação dos seus elementos:

```
carros.push("corola") //Adiciona um novo elemento no final  
frutas.pop()          //Remove do fim  
carros.shift()        //Remove do início  
frutas.unshift("Limão") //Adiciona no início
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ARRAYS:**

- Função ***splice***:

```
carros.splice(0,2) //Remove dois elemento a partir posição 0
```

```
//Adiciona os dois elementos a partir da posição 2
```

```
frutas.splice(2,1,"corola","onix") //substitui 1 elemento
```

```
frutas.splice(2,0,"corola","onix") //Não substitui
```

- Função ***sort***:

```
let numeros = [10, 2, -3, 4, 50]
```

```
numeros.sort((a,b)=>a-b) //Reordena do menor ao maior
```

Negativo: menor | **Positivo: maior** | **(a,b)=>b-a): maior ao menor**

- Função ***reverse***: `numeros.reverse()` //Apenas inverte, não ordena

- Exemplos: `g11l-jsarrays`



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO ARRAYS:**
- ***forEach()***: ou ***paraCada*** elemento, recebe uma função que é executada para cada elemento.
- A função ***callback*** recebe como parâmetro o próprio elemento da iteração corrente

```
carros.forEach(imprimeCarros);  
function imprimeCarros(carro) {  
    console.log(carro) //Imprime cada elemento  
    do array no console  
}
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO ARRAYS:**

- ***find()***: retorna o primeiro elemento encontrado dentro do array de acordo com um teste.
 - Neste caso a função **callback** deve retornar um valor **true** se o teste for verdadeiro.

```
let num = [1,2,3];
```

- `console.log(num.find(encontraPar));` //Retorna o Elemento

```
function encontraPar(n) {
```

```
    if(n%2==0) return true
```

```
}
```

- Existe também o ***findIndex()*** que retorna o índice do elemento e não o próprio elemento.
- Mais Exemplos: ***g11l-js-iter-array***



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO ARRAYS:**
- ***filter()***: retorna outro **Array** com elementos de acordo com retorna **TRUE** da função **callback**.
 - A função **callback** recebe como parâmetro o próprio elemento da iteração corrente e deve implementar um teste e retornar **true**

```
let num = [1, 2, 3, 4, 5, 6]
```

```
let impares = num.filter(filtraImpares); //Retorna um novo array
```

```
function filtraImpares(n){
```

```
    if(n%2!=0) return true
```

```
}
```

```
//Simplificando com arrow function
```

```
let impares = num.filter(n=>n%2!=0);
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO ARRAYS:**
- ***map()***: *retorna* um **novo array** como o método ***filter*** porém baseado nos retornos do ***callback***.

```
console.log(num.map(n=>n*2));
```

- ***reduce()***: retorna um único valor baseado em operações realizadas com os itens do array, por exemplo retornar a soma do total de elementos de um array.

```
let numeros = [10, 2, -3, 4, 50]
console.log(numeros.reduce((soma, item)=>soma+=item))
//Retorna a soma igual a 63
```

– Mais Exemplos: ***g11l-js-iter-array***



Desenvolvimento Front-End II

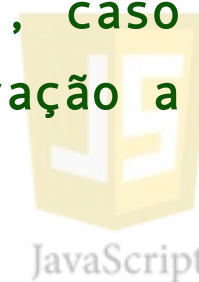
Revisão Tópicos de Javascript



- **ITERANDO ARRAYS:**

- ***reduce()***: `numeros.reduce((soma, item, indice, numeros) => soma += item, 0)`

-
- **Callback**: a função executada para cada item do array com os parâmetros a seguir.
 - **Acumulador**: o valor único a ser modificado e retornado a cada iteração
 - **Valor atual**: o valor de cada item do array durante as iterações
 - **Indice**: argumento opcional, representa o índice de cada item do array
 - **Array**: argumento opcional, o próprio array na íntegra
 - **Valor inicial**: valor a ser atribuído ao acumulador na primeira iteração, caso não seja definido, o primeiro item é utilizado, sendo realizada uma iteração a menos.
 - Veja a **documentação**. Mais exemplos de arrays: ***g11l-js-iter-array***



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **OBJETOS:**

- Em **JS** a **Orientação a Objetos** é baseada em protótipos;
- Todo o objeto possui a propriedade ***prototype*** (**`__proto__`**), ou seja, o protótipo do objeto;
- **Criando um objeto *pessoa* com atributos de altura e peso:**

```
const  pessoa = {altura: 1.7, peso: 80}
```

```
console.log({pessoa})
```

```
//Mostra o objeto pessoa no console
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript

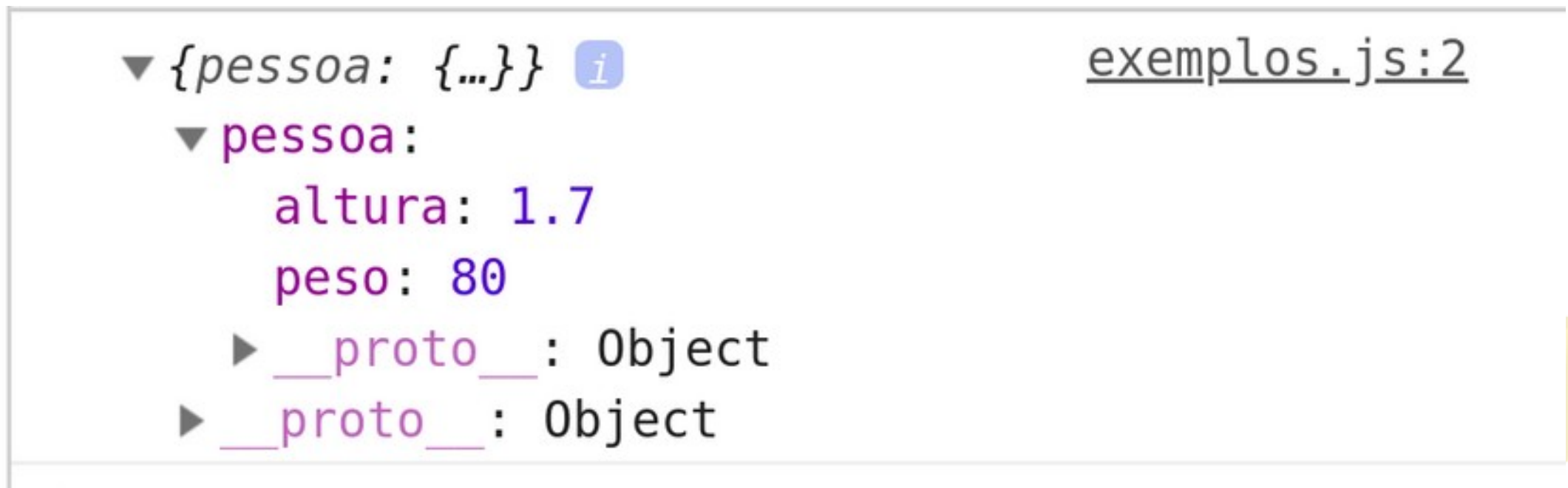


- **OBJETOS:**

- Exemplos

Console browser: `{ pessoa: {altura: 1.7, peso: 80 } }`

Abrir console: *ctrl+shift+j*



javascript

Link: Referência de Objetos Globais

Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **OBJETOS:**

- **Exemplo no Node:**

```
let pessoa = { altura: 1.7, peso: 80 }
```

```
> let pessoa = { altura: 1.7, peso: 80 }  
undefined  
  
> pessoa  
{ altura: 1.7, peso: 80 }
```

- Links de *Exemplos*
 - **DOCS: MAIS SOBRE OBJECT**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **OBJETOS:**

- É possível **adicionar** novos **atributos** a um **objeto** já **declarado**
- Por exemplo **adicionar** o atributo **idade** no objeto **pessoa**:

```
const  pessoa = {altura: 1.7, peso: 80}
```

```
console.log({pessoa}) //Mostra objeto sem idade
```

```
pessoa.idade = 31 //Atributo adicionado ao objeto
```

```
console.log({pessoa}) //Objeto com idade
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **OBJETOS:**

- É possível **acessar** atributos ***dinamicamente***, ou programáticamente.
- Utiliza-se uma ***variável*** entre colchetes **[]** após o objeto.
- Por exemplo, mostrar uma informação específica do objeto ***carro*** de acordo com a variável ***mostrar***:

```
const  carro = {marca: 'VW', cor: 'Preto', idade: 0}  
let  mostrar = 'cor';  
console.log(`A ${mostrar} do carro é ${carro[mostrar]}`);  
//saída: A cor do carro é Preto
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **OBJETOS:**

- Mesmo sendo suportado não é recomendado, pois dificulta a leitura de quais são as propriedades do objeto;
- É melhor já criar o atributo idade na declaração, mesmo que depois seu valor seja modificado;
- Nestes casos também é mais interessante trabalhar com **Classes**.
- Em **versões antigas** eram utilizadas **functions** para simular **POO**.
- Mas a **versão 2015 (ES6)**, já possui sintaxe própria de **POO**



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO OBJETOS:**

- É possível iterar atributos de um objeto usando o método ***entries()*** que retorna um par **[chave, valor]** para cada propriedade do Objeto.
- Nos casos abaixo temos acesso aos nomes e aos valores dos atributos do objeto como ***arrays***.

```
const carro = {marca: 'VW', cor: 'Preto', idade: 0}
//Mostra arrays para cada par chave, valor do objeto
for (let itens of Object.entries(carro)) {
    console.log(itens); //Item é um array [chave, valor]
}
```



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO OBJETOS:**

- Exemplo do código anterior no *nodejs*:

```
> Object.entries(carro)
[ [ 'marca', 'VW' ], [ 'cor', 'Preto' ], [ 'idade', 0 ] ]
> for(let item of Object.entries(carro)){ console.log({item})}
{ item: [ 'marca', 'VW' ] }
{ item: [ 'cor', 'Preto' ] }
{ item: [ 'idade', 0 ] }
```

- Cada item na imagem acima representa um atributo do objeto com seu nome e valor.
 - São três itens pois o objeto tem também três atributos, gerando um **array** de três elementos pelo método *entries()*.



Desenvolvimento Front-End II

Revisão Tópicos de Javascript



- **ITERANDO OBJETOS:**

- Acessar diretamente a **chave** e o **valor** de cada **atributo** do objeto:

//Neste caso temos acesso direto a chave e valor de cada atributo do objeto em cada iteração do FOR(OF)

```
for (let [chave, valor] of Object.entries(carro)) {  
    console.log(`atributo: ${chave} | valor: ${valor}`);  
}
```

- Só é possível por que o método entries da classe Object retorna um **Array** (**iterável**), contendo **elementos** que representam os **atributos do objeto** como pares **chave-valor**, sendo a **chave** o **nome do atributo**.



Desenvolvimento Front-End II

Revisão Tópicos de Javascript

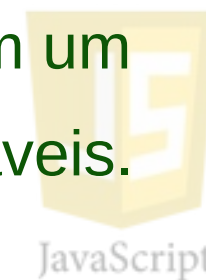


- **ITERANDO OBJETOS:**

- Exemplo executado no *nodejs*:

```
> Object.entries(carro)
[ [ 'marca', 'VW' ], [ 'cor', 'Preto' ], [ 'idade', 0 ] ]
> for(let [k,v] of Object.entries(carro)){ console.info(`atributo: ${k}\t| valor: ${v}`)}
atributo: marca | valor: VW
atributo: cor   | valor: Preto
atributo: idade | valor: 0
```

- Exemplos: `stackblitz` `gill-iterobjects`
- Tudo Isso funciona por que o tipo array é considerado um iterável, também um tipo especial e novo no **JS**, onde podemos criar o nosso próprios tipo iteráveis.
- Docs: Mais sobre iteradores e geradores



Desenvolvimento Front-End II

Ferramentas de desenvolvimento Front-End



- **Bundlers (Empacotadores)**

- Com o ***aumento da complexidade*** de aplicações o *Javascript* se tornou **modularizado**, a começar com os pacotes *node* e *npm*.
- ***Browserify*** foi o **primeiro** a dar suporte a códigos modularizados aos navegadores, criando um pacote com todas as dependências necessárias.
- Normalmente utilizado com ***Gulp/Grunt*** também para **preprocessadores** de **CSS** como o **SASS**.
- Existem ainda outro bundlers como ***Webpack*** ou ***Parcel***.
- O ***Rollup*** também é um **bundler** relativamente fácil de configurar, semelhante ao antigo *Browserify*.
- Usado como padrão em frameworks como o ***Svelt***.
- E ainda existe o ***Vite*** (/vit/), com uma proposta diferente e prometendo ser mais rápido do que os anteriores por aproveitar o ***Native ESM***.



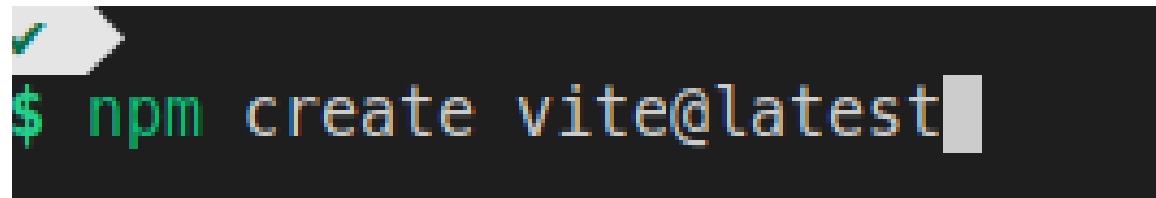
Desenvolvimento Front-End II

Usando Ferramentas com suporte ao EcmaScript Module com Native ESM



Vite

- ▶ É uma ferramenta de desenvolvimento baseada no suporte a módulos no navegador (**Native ESM**).
- ▶ Sem a necessidade de empacotar (*bundling*) todo o código javascript.
- ▶ Mais rápido para o desenvolvimento em comparação com *bundlers* anteriores.
- ▶ Separa a código-fonte de dependências, melhorando o carregamento do servidor de desenvolvimento.
- ▶ Para criar um projeto usamos o comando **npm**:

A screenshot of a terminal window with a dark background. It shows a command prompt '\$' followed by the text 'npm create vite@latest' in a monospaced font. A small grey cursor is at the end of the command. There is a small icon in the top left corner of the terminal window.

- ▶ ➔ **Documentação**

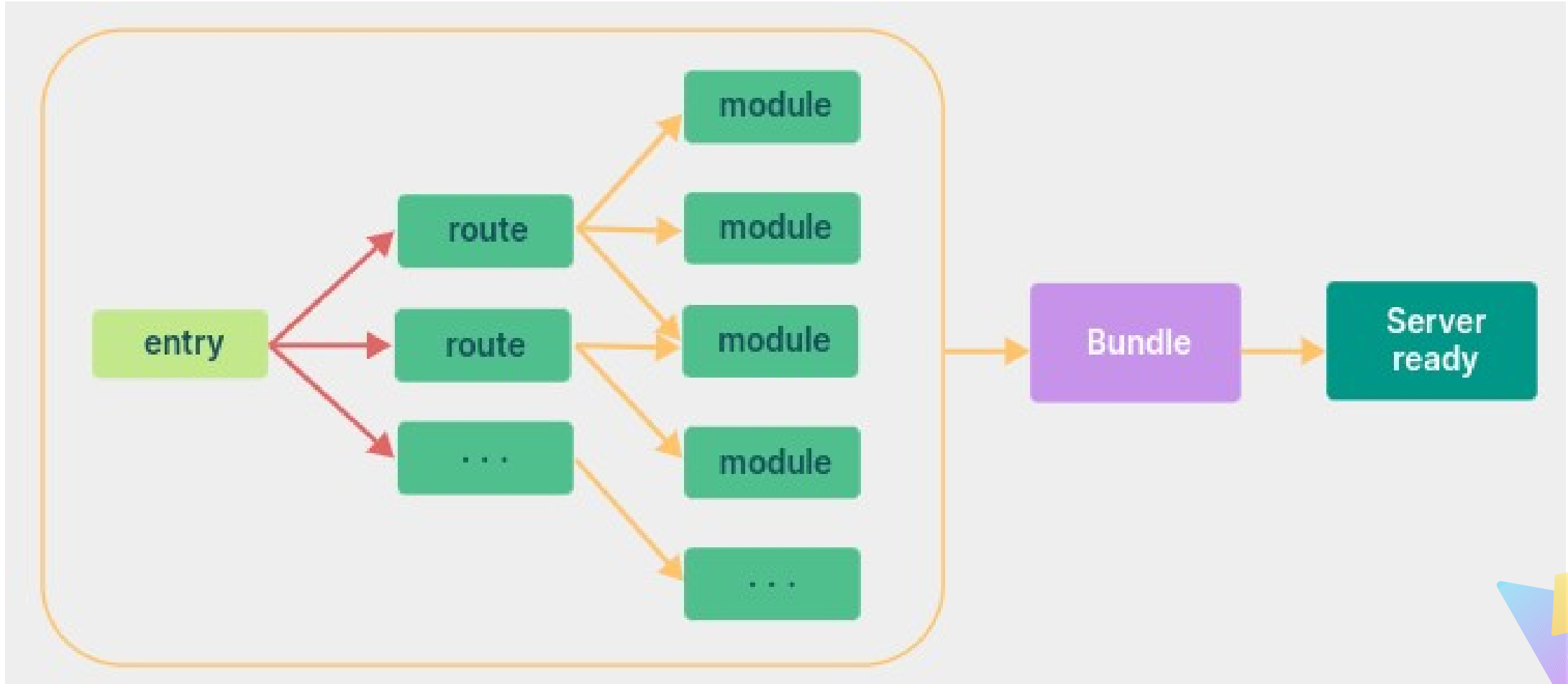
Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Native ESM x Bundler

- **Bundler:**



► ➡ Documentação

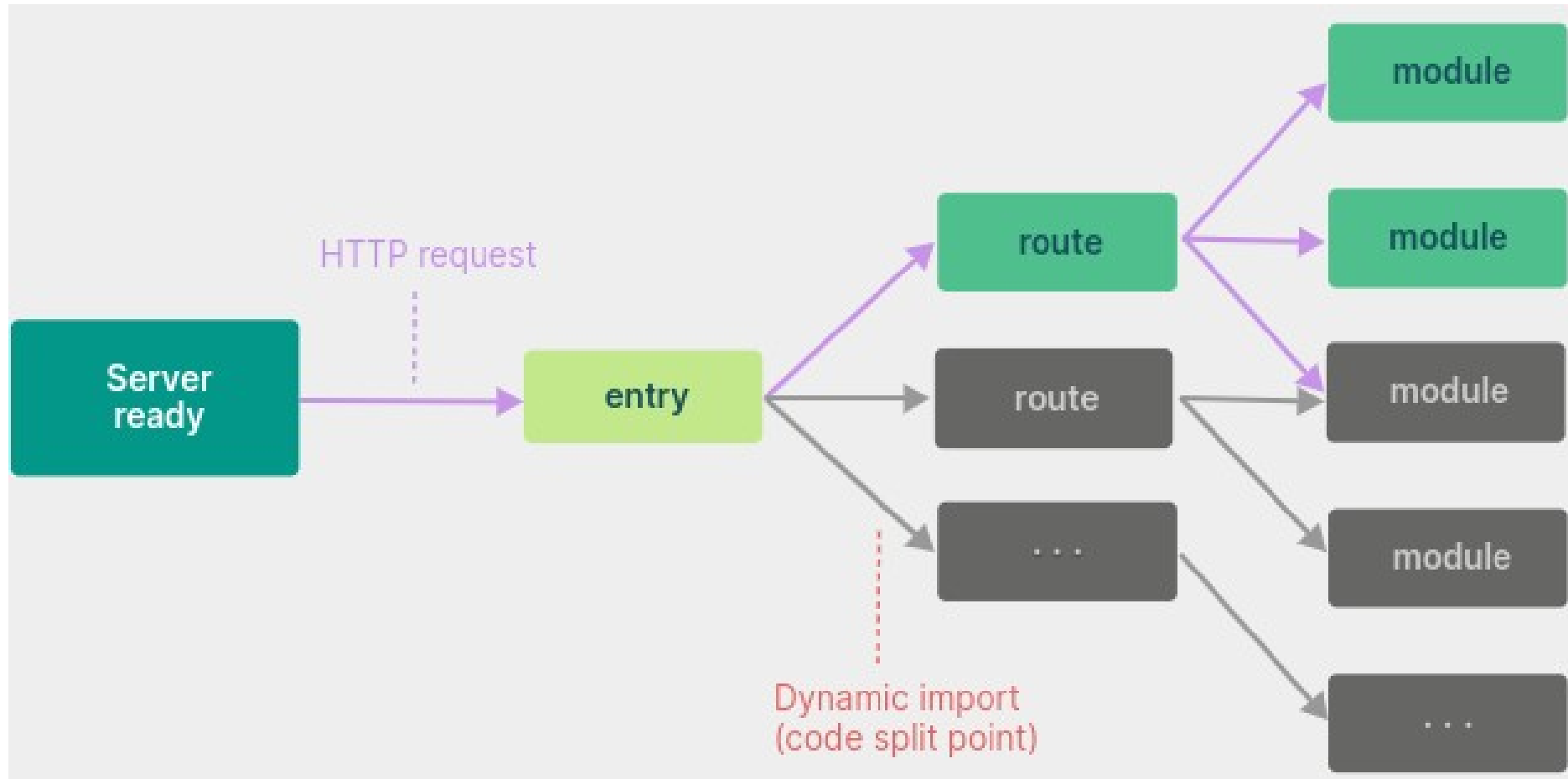
Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Native ESM x Bundler

- **ESM:**



► Documentação



Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Vite

- ▶ Exemplo de criação de projeto com *npm* ou *yarn*:

NPM	Yarn	PNPM	Bun
<pre>\$ npm create vite@latest</pre>			

- ▶ ➡ [Link Doc](#)

NPM	Yarn	PNPM	Bun
<pre>\$ yarn create vite</pre>			



Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Vite

- ▶ Será perguntado sobre o nome do projeto, na verdade este será também o nome da pasta ao qual o vite criará para o projeto.
- ▶ Após podemos escolher qual *framework front-end* trabalhar, neste exemplo usarei *vanilla JS*.

```
$ npm create vite@latest
✓ Project name: ... exemplo-41-vite-setup
? Select a framework: > - Use arrow-keys. Return to submit.
>  vanilla
   vue
   react
   preact
   lit
   svelte
```

- ▶ Não segunda pergunta **não** escolha *vanilla-ts*, pois é para **TypeScript**.



Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Vite

- ▶ Após os comandos de criação, teremos na pasta com o nome do projeto os arquivos iniciais como a configuração do npm ***package.json***.
- ▶ Entramos na pasta com ***cd*** e rodamos os comandos de instalação de dependências do vite e execução de um servidor de desenvolvimento.

- ▶ ➡ ***Link Documentação***

```
exemplo-41-vite-setup...
```

```
Done. Now run:
```

```
cd exemplo-41-vite-setup  
npm install  
npm run dev
```

Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Vite

- ▶ Também é possível criar o projeto e escolher o *template* diretamente com o seguinte comando:

```
NPM  Yarn  PNPM  Bun
$ yarn create vite my-vue-app --template vue
```

See [create-vite](#) for more details on each supported template: `vanilla`, `vanilla-ts`, `vue`, `vue-ts`, `react`, `react-ts`, `react-swc`, `react-swc-ts`, `preact`, `preact-ts`, `lit`, `lit-ts`, `svelte`, `svelte-ts`, `solid`, `solid-ts`, `qwik`, `qwik-ts`.

- ▶ ➡ [Link Doc](#)

Criando um projeto vanilla (js puro):

- ▶ `yarn create vite myApp --template vanilla`
- ▶ `npm create vite@latest myApp -- --template vanilla`



Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



YARN:

- *depois é necessário entrar na pasta do projeto:*

- ▶ `cd myApp`
- ▶ `yarn`
- ▶ `yarn dev`

NPM:

- *depois é necessário entrar na pasta do projeto:*

- ▶ `cd myApp`
- ▶ `npm install`
- ▶ `npm run dev`

```
Projetos yarn create vite myApp --template vanilla
yarn create v1.22.21
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Installed "create-vite@5.5.2" with binaries:
  - create-vite
  - cva
✓ Package name: ... myapp

Scaffolding project in D:\Projetos\myApp...

Done. Now run:

  cd myApp
  yarn
  yarn dev

Done in 6.88s.

gill@D9320 > Projetos > ✓
```


Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



dev:

- O comando **dev** iniciará o servidor de desenvolvimento com o **hot-reload** configurado, facilitando a atualização da página durante o desenvolvimento.
- Exemplos das aulas com vite:
 - entrar na pasta **exemplo**
 - rodar os **comandos do vite**
- Faça download das aulas:
⇒ **repositório da aula**
- Ou use o stackblitz:
⇒ **stackblitz da aula**

```
gill@D9320 > ctsi-4sem-dfe2 > main ≡ ?3 -27 > ✓
> cd .\aula-01-vite-css3-js\exemplo-01-01-vite\

exemplo-01-01-vite > npm install

up to date, audited 13 packages in 8s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

gill@D9320 > exemplo-01-01-vite > main ≡ ?3 -27 > ✓
> npm run dev
```

Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



dev:

- *O servidor de desenvolvimento estará disponível na porta **5173** no **localhost**.*

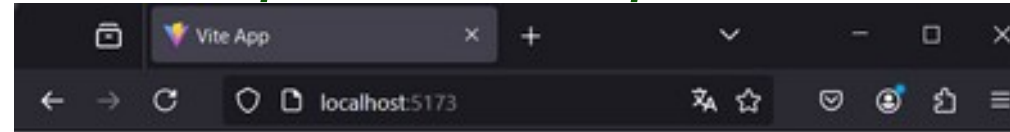
```
exemplo-01-01-vite npm run dev

> exemplo-01-01@0.0.0 dev
> vite

Re-optimizing dependencies because vite

VITE v5.4.8 ready in 839 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```



Hello Vite!

count is 0

Click on the Vite logo to learn more

Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Linux: erro EMFILE

- Pode aparecer este erro ao iniciar o servidor de desenvolvimento, isso ocorre porque o vite precisa monitorar diversos arquivos para entregar eles separadamente ao navegador de acordo com as requisições.*

```
> react@0.0.0 dev
```

```
> vite
```

```
node:internal/fs/watchers:247
```

```
  const error = new UVException({  
    ^
```

```
Error: EMFILE: too many open files, watch '/media/Programas/Projetos/t
```

Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



Linux: erro EMFILE

- *Para corrigir este erro é preciso configure o seu linux e permitir que o vite monitore um numero maior de arquivos.*
- *Execute os seguintes comandos no seu terminal:*

```
# Check current limits
$ sysctl fs.inotify
# Change limits (temporary)
$ sudo sysctl fs.inotify.max_queued_events=16384
$ sudo sysctl fs.inotify.max_user_instances=8192
$ sudo sysctl fs.inotify.max_user_watches=524288
```

shell

► ➡ ***Link Documentação***

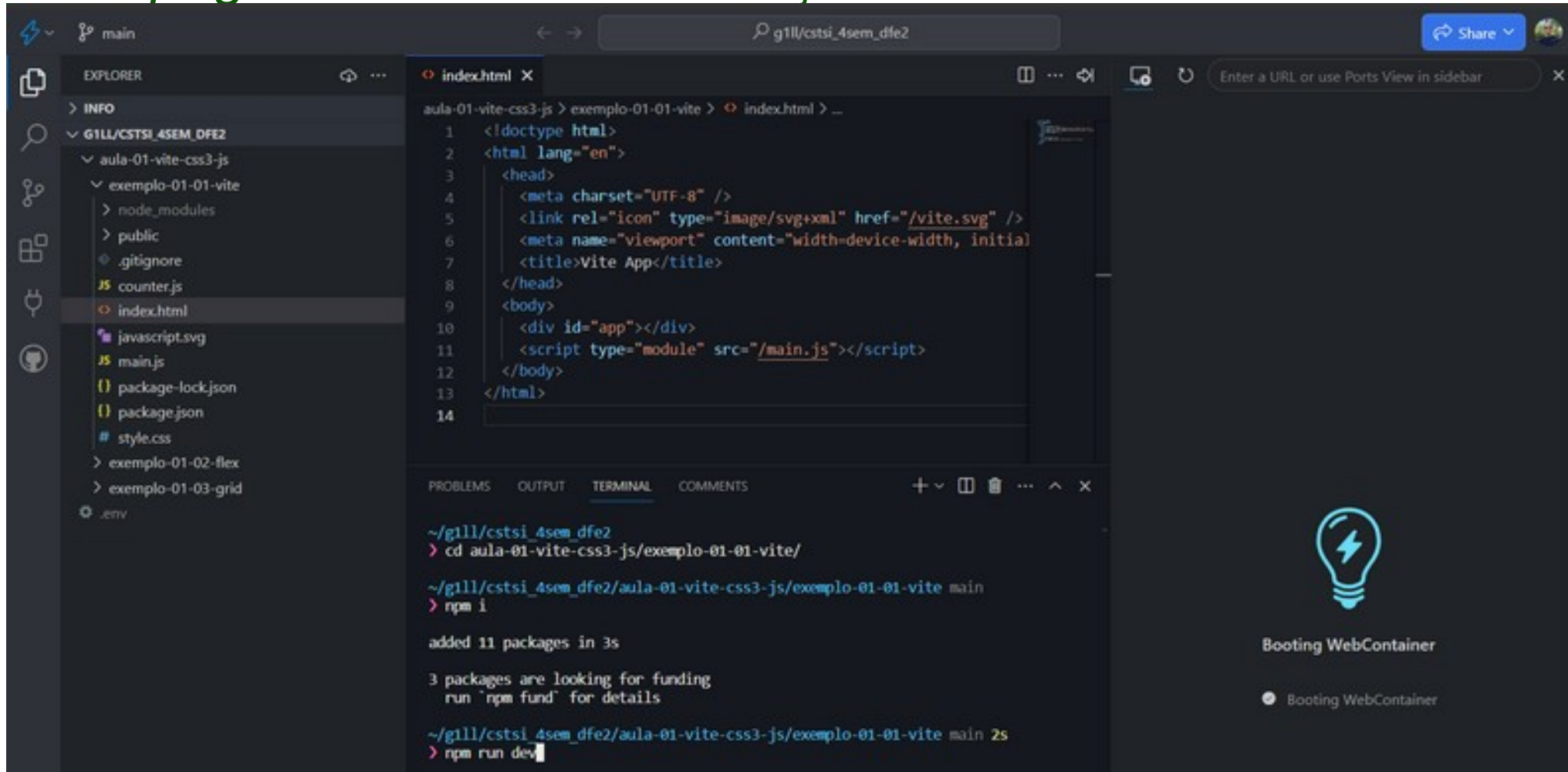
Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



*Você pode rodar os mesmos comandos no terminal da plataforma **stackblitz**, mas a página será aberta na seção ao lado.*

➤ ➡ **exemplo**

A screenshot of a web-based code editor interface. The left sidebar shows a file explorer with a project structure: 'aula-01-vite-css3-js' > 'exemplo-01-01-vite'. The main editor area shows the 'index.html' file with the following content:

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6     <meta name="viewport" content="width=device-width, initial
7   <title>Vite App</title>
8 </head>
9 <body>
10  <div id="app"></div>
11  <script type="module" src="/main.js"></script>
12 </body>
13 </html>
14
```

The bottom panel shows a terminal with the following commands and output:

```
~/gill/cstsi_4sem_dfe2
> cd aula-01-vite-css3-js/exemplo-01-01-vite/

~/gill/cstsi_4sem_dfe2/aula-01-vite-css3-js/exemplo-01-01-vite main
> npm i

added 11 packages in 3s

3 packages are looking for funding
  run "npm fund" for details

~/gill/cstsi_4sem_dfe2/aula-01-vite-css3-js/exemplo-01-01-vite main 2s
> npm run dev
```

On the right side of the terminal, there is a lightbulb icon and the text 'Booting WebContainer'.

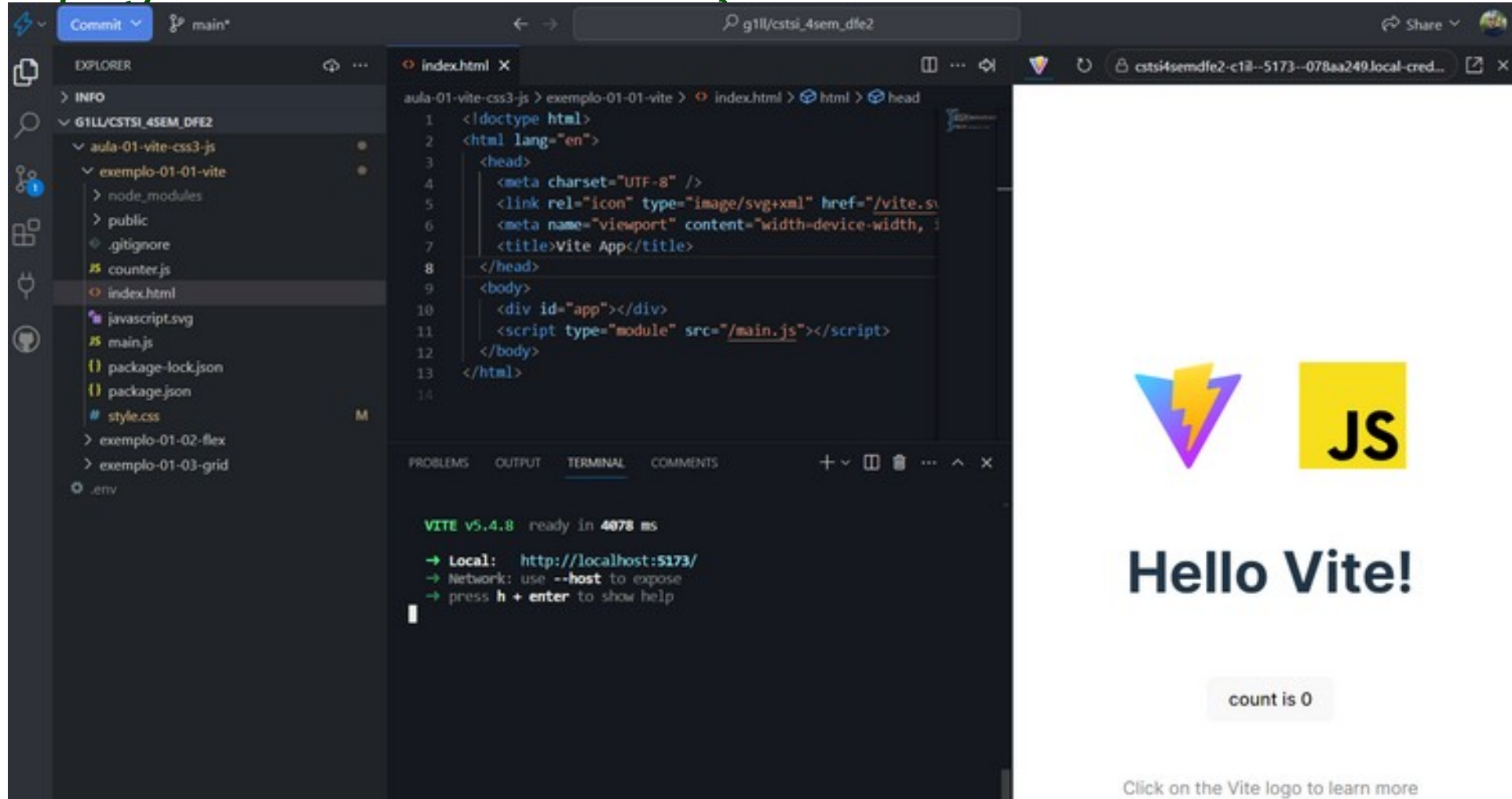
Native ESM

Usando Ferramentas com suporte ao EcmaScript Module



*Você pode rodar os mesmos comandos no terminal da plataforma **stackblitz**, mas a página será aberta na seção ao lado.*

► ➡ **exemplo**





REFERÊNCIAS

SILVA, Maurício Samy. React Aprenda Praticando. São Paulo: Novatec, 2021.

Getting Started – React: 2022 Meta Platforms, Inc. Disponível em: <https://reactjs.org/docs/getting-started.html>. Acesso em: 17/10/2022.

Documentation: Tailwind CLI - Tailwind CSS. 2022 Tailwind Labs Inc. Disponível em: <https://tailwindcss.com/docs/installation>. Acesso em: 17/10/2022.

SILVA, Maurício Samy. CSS3 – Desenvolva aplicações web profissionais com o uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec, 2011.

SILVA, Maurício Samy. HTML5 – A linguagem de marcação que revolucionou a web. São Paulo: Novatec, 2011.

SILVA, Maurício Samy. JavaScript: guia do programador. São Paulo: Novatec, 2010.



REFERÊNCIAS

https://www.w3schools.com/js/js_htmlDOM.asp

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects

<https://www.alura.com.br/artigos/entenda-diferenca-entre-var-let-e-const-no-javascript>

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object

https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>