



MÓDULOS / WEBPACK

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

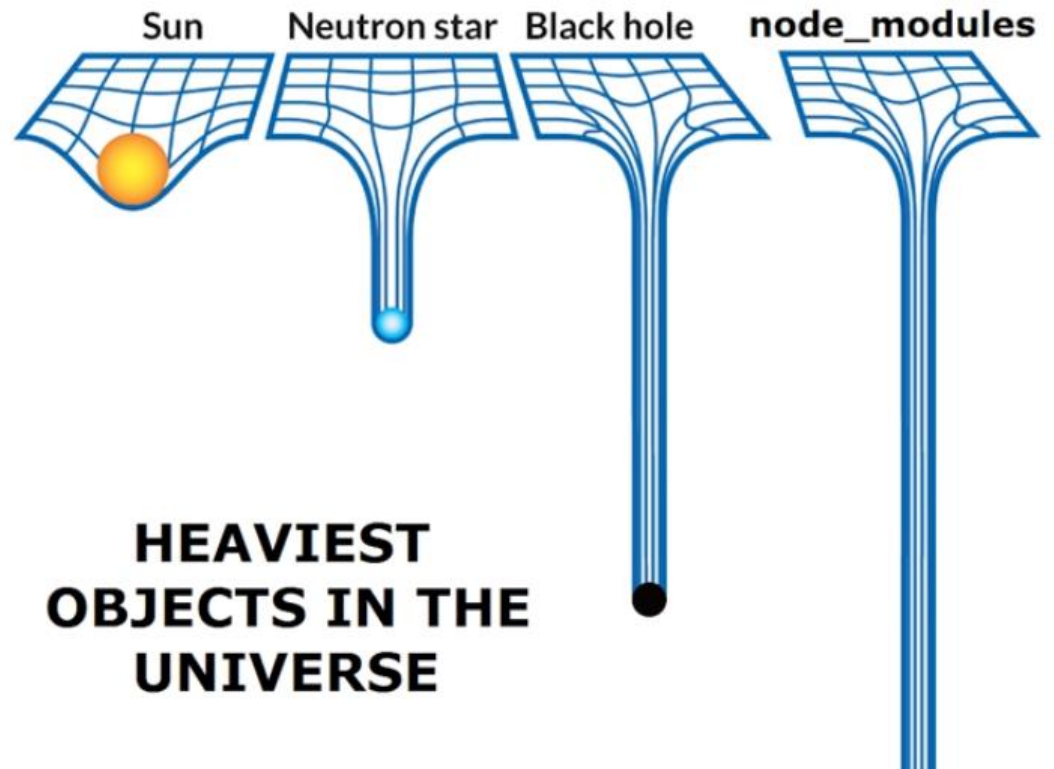
HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

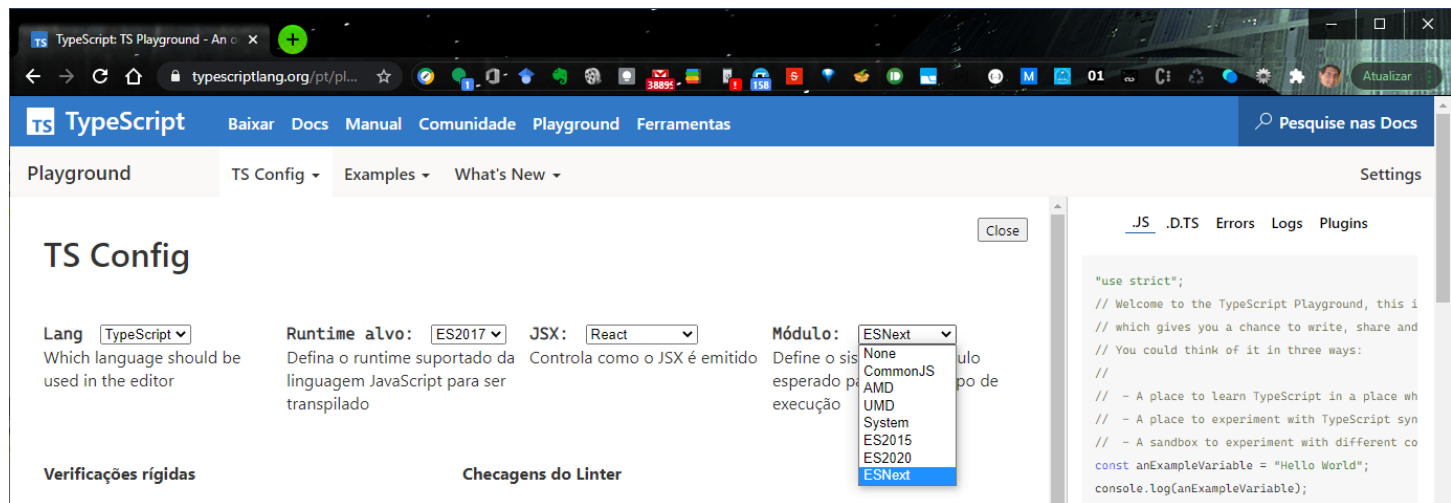
- Módulos
- Webpack



MÓDULOS

■ Sistemas de módulos em JS

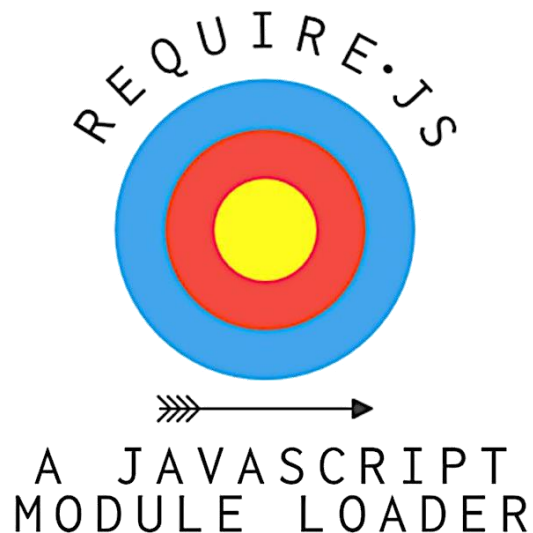
- Padrão [ECMAScript 2015/ES6](#) (ESM): Utiliza as instruções `import/export`
- [CommonJS](#) (CJS): Criado para estabelecer convenções do ecossistema de módulos para o JavaScript fora do navegador, é utilizado no Node.js e utiliza `export` e `require`
- [AMD](#) (*Asynchronous Module Definition*): Alternativa para a especificação CommonJS, permite o carregamento assíncrono dos módulos e suas dependências
- [UMD](#) (*Universal Module Definition*): Padrão para definição universal de módulos no JS que são capazes de rodar em qualquer lugar (cliente, servidor, etc), oferecendo compatibilidade com os *loaders* mais populares; usa AMD como base



MÓDULOS

- Carregadores (loaders)

- RequireJS: Carregador de módulos otimizado para uso em navegadores



- webpack: Bundler de pacotes/módulos



MÓDULOS

- As três formas abaixo usam diferentes objetos para exportarem recursos
 - `this:` Aponta para o contexto local
 - `exports:` Utilizado para exportar entidades, torna os objetos disponíveis fora do módulo
 - `module:` Exporta um objeto completo
- Para realizar a importação deve-se usar as instruções `require` ou `import`

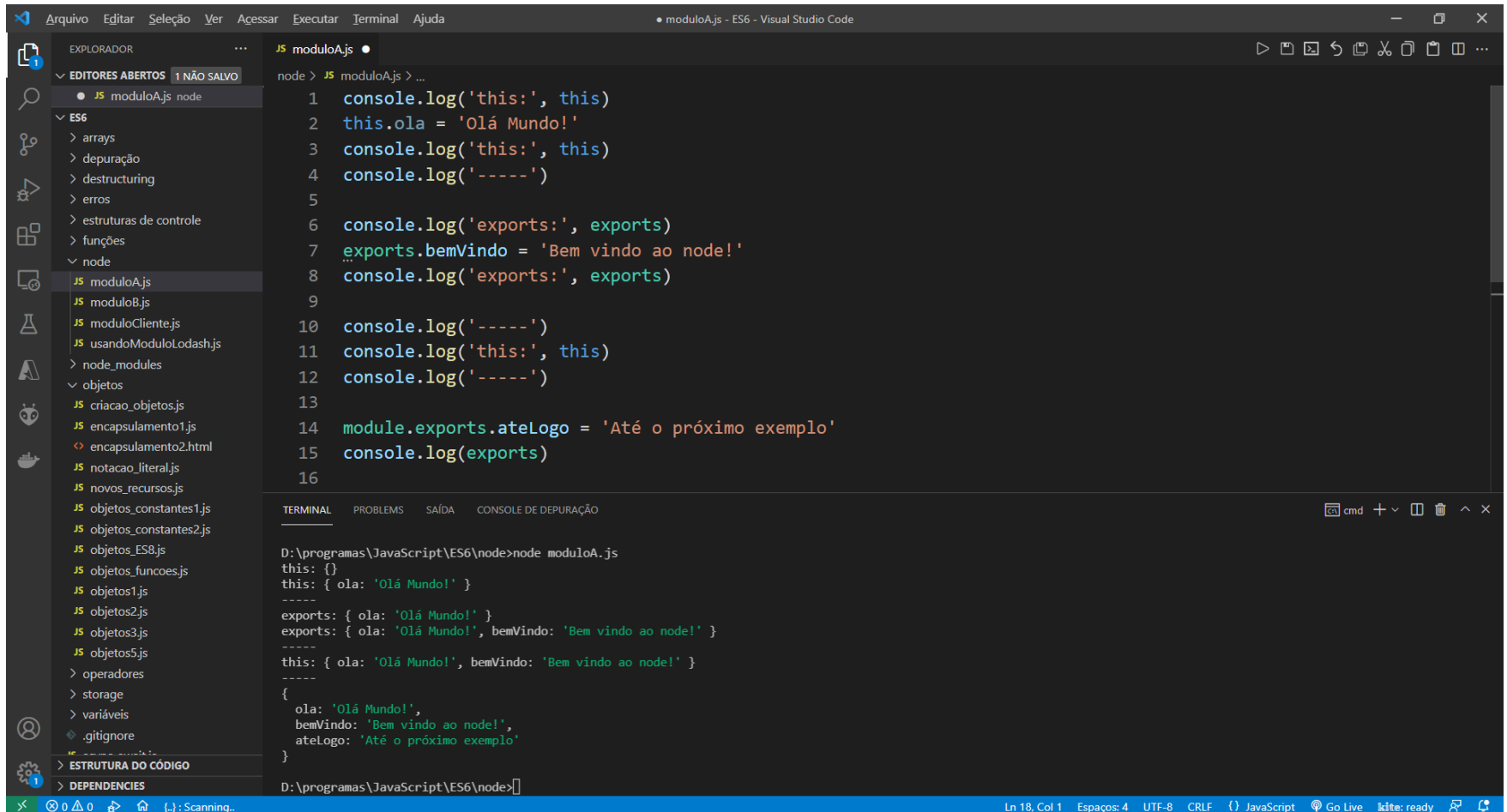
MÓDULOS

■ Diferenças entre os módulos do CommonJS e o ES6

- O `require` existe só em CommonJS (é a maneira que o Node.js criou para importar e exportar módulos), e o `import` é ES6, ou seja uma nova ferramenta que ambos JavaScript do navegador e JavaScript do servidor (Node.js) podem usar
- O `import` é mais flexível, moderno e poderoso que o `require`
- O `import` pode usar tanto o `module.exports` quanto o `export`, e permite exportar vários pedaços de código, mais ou menos como o `module.exports` fazia
- Uma das vantagens do `import` é poder importar só partes do que foi exportado
- O `require` usa `module.exports`, que é a sintaxe “antiga” (mas ainda válida) para exportar um módulo, e que pode ser o que quisermos, um objeto, uma *string*, etc.

MÓDULOS

CommonJS (Node): Módulo A mostrando as diferentes formas de exportação



The screenshot displays the Visual Studio Code editor with a file named `moduloA.js` open. The file contains the following code:

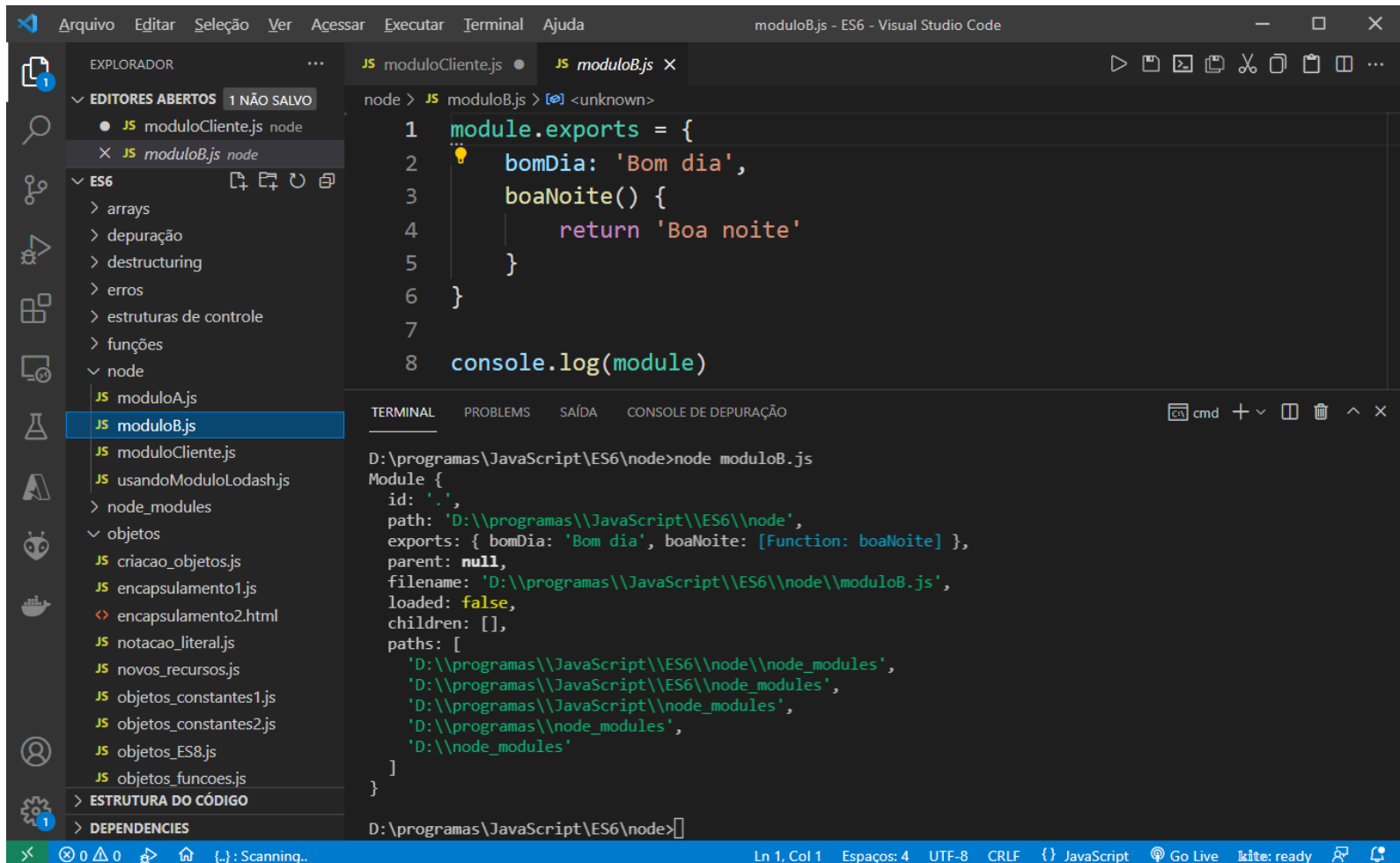
```
node > JS moduloA.js > ...
1 console.log('this:', this)
2 this.ola = 'Olá Mundo!'
3 console.log('this:', this)
4 console.log('-----')
5
6 console.log('exports:', exports)
7 exports.bemVindo = 'Bem vindo ao node!'
8 console.log('exports:', exports)
9
10 console.log('-----')
11 console.log('this:', this)
12 console.log('-----')
13
14 module.exports.ateLogo = 'Até o próximo exemplo'
15 console.log(exports)
16
```

The terminal window at the bottom shows the output of running `node moduloA.js`:

```
D:\programas\JavaScript\ES6\node>node moduloA.js
this: {}
this: { ola: 'Olá Mundo!' }
-----
exports: { ola: 'Olá Mundo!' }
exports: { ola: 'Olá Mundo!', bemVindo: 'Bem vindo ao node!' }
-----
this: { ola: 'Olá Mundo!', bemVindo: 'Bem vindo ao node!' }
-----
{
  ola: 'Olá Mundo!',
  bemVindo: 'Bem vindo ao node!',
  ateLogo: 'Até o próximo exemplo'
}
```

MÓDULOS

■ CommonJS (Node): Módulo *B* mostrando a forma mais clássica de exportação



The image shows a Visual Studio Code editor window with the file `moduloB.js` open. The file contains the following JavaScript code:

```
1 module.exports = {  
2   bomDia: 'Bom dia',  
3   boaNoite() {  
4     return 'Boa noite'  
5   }  
6 }  
7  
8 console.log(module)
```

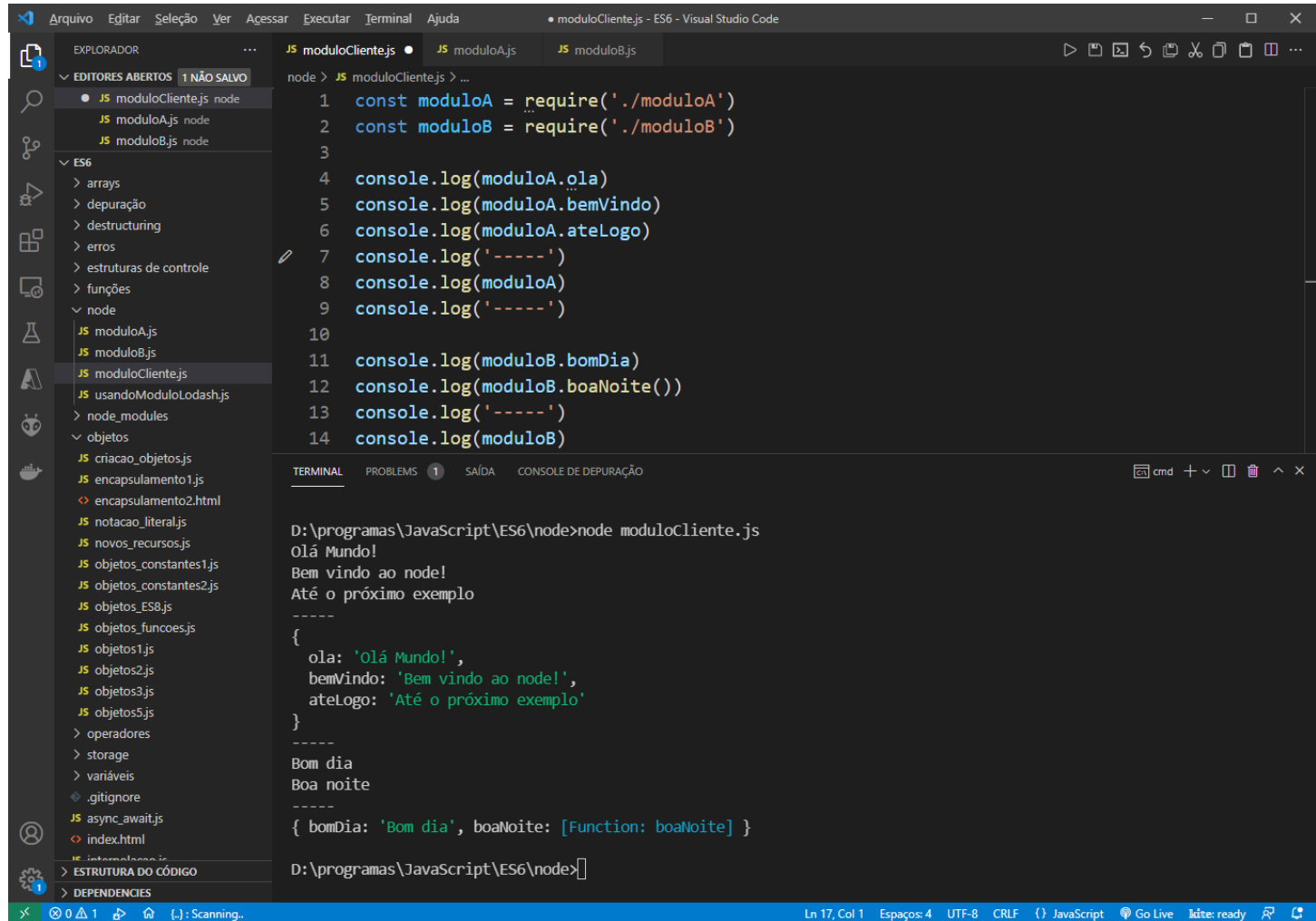
The left sidebar shows the Explorer view with the file structure. The file `moduloB.js` is selected under the `node` folder. The bottom panel shows the Terminal with the command `node moduloB.js` executed, resulting in the following output:

```
D:\programas\JavaScript\ES6\node>node moduloB.js  
Module {  
  id: '.',  
  path: 'D:\\programas\\JavaScript\\ES6\\node',  
  exports: { bomDia: 'Bom dia', boaNoite: [Function: boaNoite] },  
  parent: null,  
  filename: 'D:\\programas\\JavaScript\\ES6\\node\\moduloB.js',  
  loaded: false,  
  children: [],  
  paths: [  
    'D:\\programas\\JavaScript\\ES6\\node\\node_modules',  
    'D:\\programas\\JavaScript\\ES6\\node_modules',  
    'D:\\programas\\JavaScript\\node_modules',  
    'D:\\programas\\node_modules',  
    'D:\\node_modules'  
  ]  
}
```

The status bar at the bottom indicates the current line and column (Ln 1, Col 1), encoding (UTF-8), and other settings.

MÓDULOS

■ CommonJS (Node): Importando os módulos *A* e *B*



The screenshot displays the Visual Studio Code editor with a project named 'moduloCliente.js - ES6 - Visual Studio Code'. The Explorer sidebar on the left shows the file structure, including 'moduloA.js', 'moduloB.js', and 'moduloCliente.js'. The main editor window shows the code in 'moduloCliente.js'.

```
node > JS moduloCliente.js > ...
1  const moduloA = require('./moduloA')
2  const moduloB = require('./moduloB')
3
4  console.log(moduloA.ola)
5  console.log(moduloA.bemVindo)
6  console.log(moduloA.atelogo)
7  console.log('-----')
8  console.log(moduloA)
9  console.log('-----')
10
11 console.log(moduloB.bomDia)
12 console.log(moduloB.boaNoite())
13 console.log('-----')
14 console.log(moduloB)
```

The terminal at the bottom shows the command `D:\programas\JavaScript\ES6\node>node moduloCliente.js` and its output:

```
D:\programas\JavaScript\ES6\node>node moduloCliente.js
Olá Mundo!
Bem vindo ao node!
Até o próximo exemplo
-----
{
  ola: 'Olá Mundo!',
  bemVindo: 'Bem vindo ao node!',
  atelogo: 'Até o próximo exemplo'
}
-----
Bom dia
Boa noite
-----
{ bomDia: 'Bom dia', boaNoite: [Function: boaNoite] }
```

The terminal prompt is `D:\programas\JavaScript\ES6\node>`.

MÓDULOS

■ ECMAScript 2015 (ES6)

- `export`: Exporta recursos

- `import`: Importa recursos

- Objetos importados podem receber um apelido

- É possível exportar vários objetos de dentro de um mesmo arquivo, mas somente um deles poderá sofrer uma exportação *default*

MÓDULOS

■ ECMAScript 2015 (ES6)

[illegible]

MÓDULOS

■ ECMAScript 2015 (ES6)

Can I use ? ⚙ Settings

10 results found

☒ Caniuse (3) ☒ MDN (7)

JavaScript statement: export

Usage: Global 95.46%

Current aligned Usage relative Date relative Filtered All ⚙

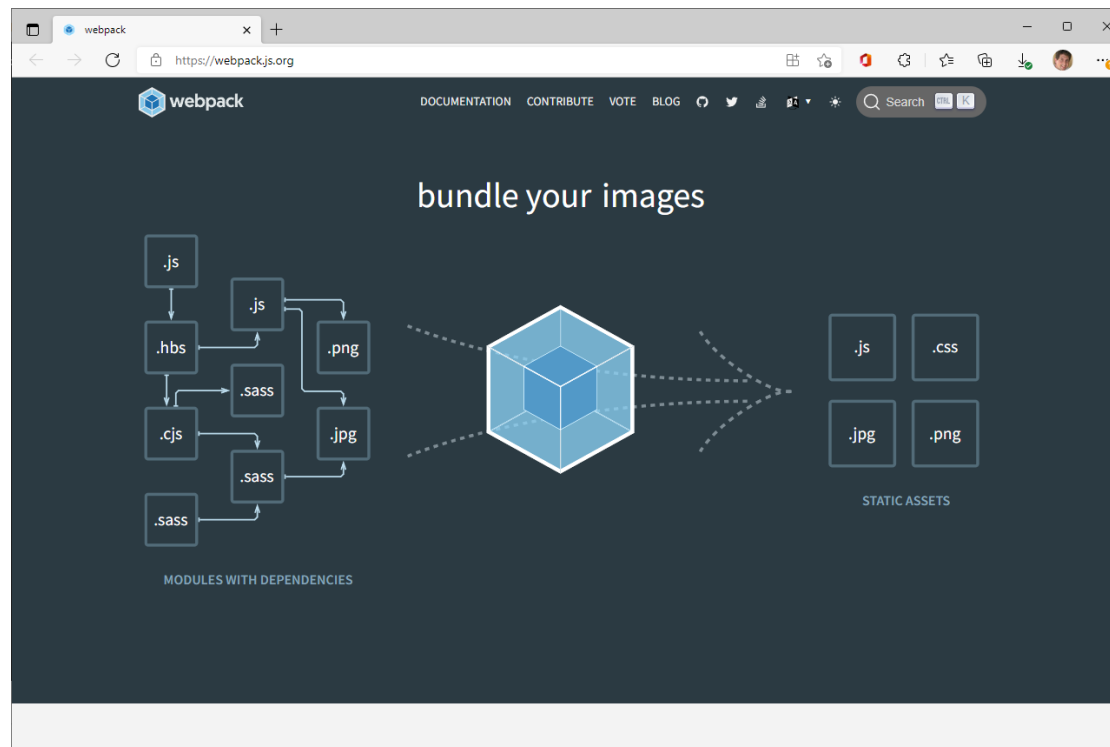
Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-60	12-15	3.1-10	2-59	10-47			3.2-10.2	4-7.4								
61-107	16-107	10.1-16.0	60-106	48-91	6-10		10.3-16.0	8.2-18.0		12-12.1		2.1-4.4.4				
108	108	16.1	107	92	11	108	16.1	19.0	all	72	13.4	108	107	13.1	13.18	2.5
109-111		16.2-TP	108-109													

Notes Test on a real browser Sub-features Feedback

See full reference on [MDN Web Docs](#).

WEBPACK

- Webpack é um empacotador (*bundler*) *open source* de módulos JavaScript (padrão CommonJS, ES2015, etc) que agrupa todo o código JavaScript e demais *assets* do *front-end* (HTML, CSS, imagens, etc) em um único arquivo com os recursos estáticos
- É um módulo do node.js, sendo executado por ele, mais informações podem ser vistas [aqui](https://webpack.js.org)
- Grande variedade de *plug-ins*



WEBPACK

- Webpack auxilia em tarefas como
 - Otimização de imagens
 - Minificação de arquivos
 - Compatibilidade entre navegadores
 - Controle de escopo
 - Redução da quantidade de requisições feitas pela aplicação
 - Gerenciamento de dependências