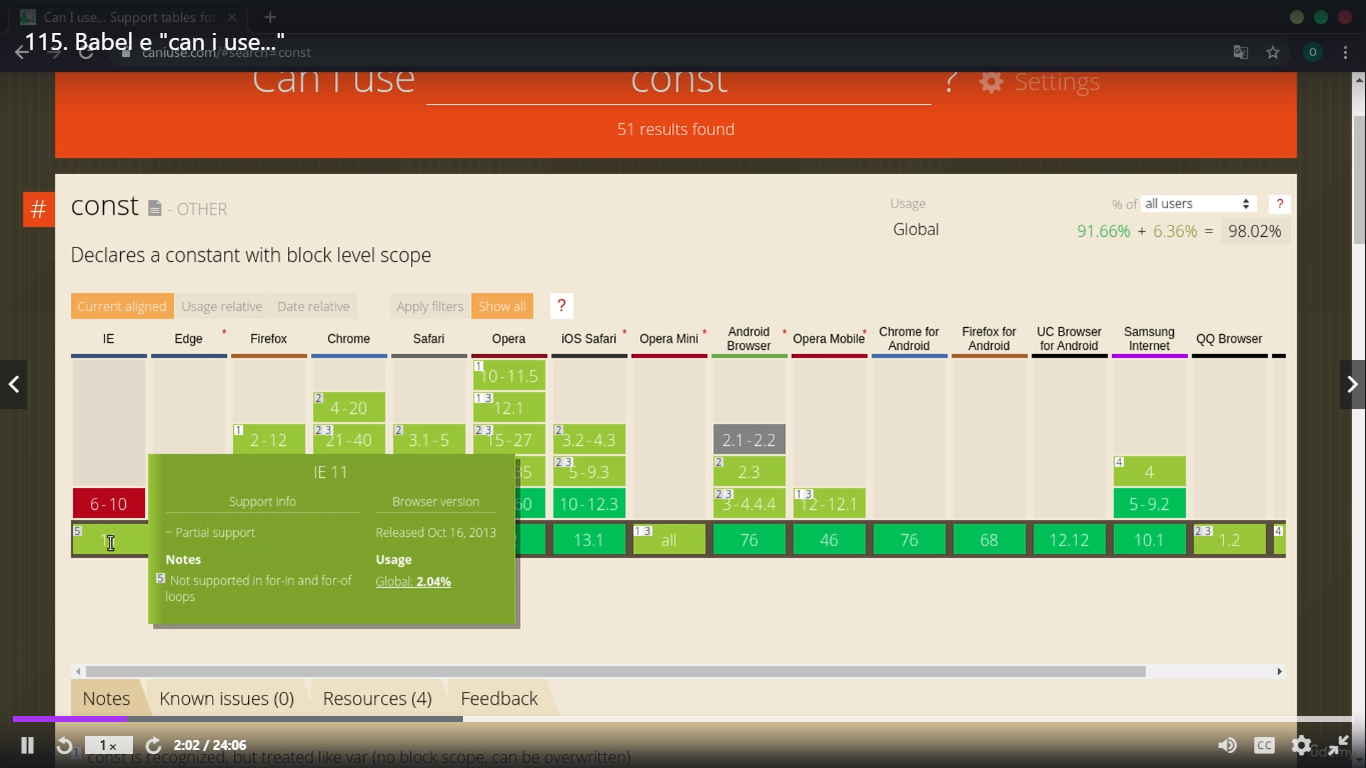
Seção 09 Javascript Tooling e ES6 Modules – Módulos

Se você pegar um estatitica de uso de navegadores, você vera que o internet explore, ainda tem sua representatividade nos usos de navegadores, e muitos dos recursos que aprendemos não são suportados por esse e alguns navegadores.

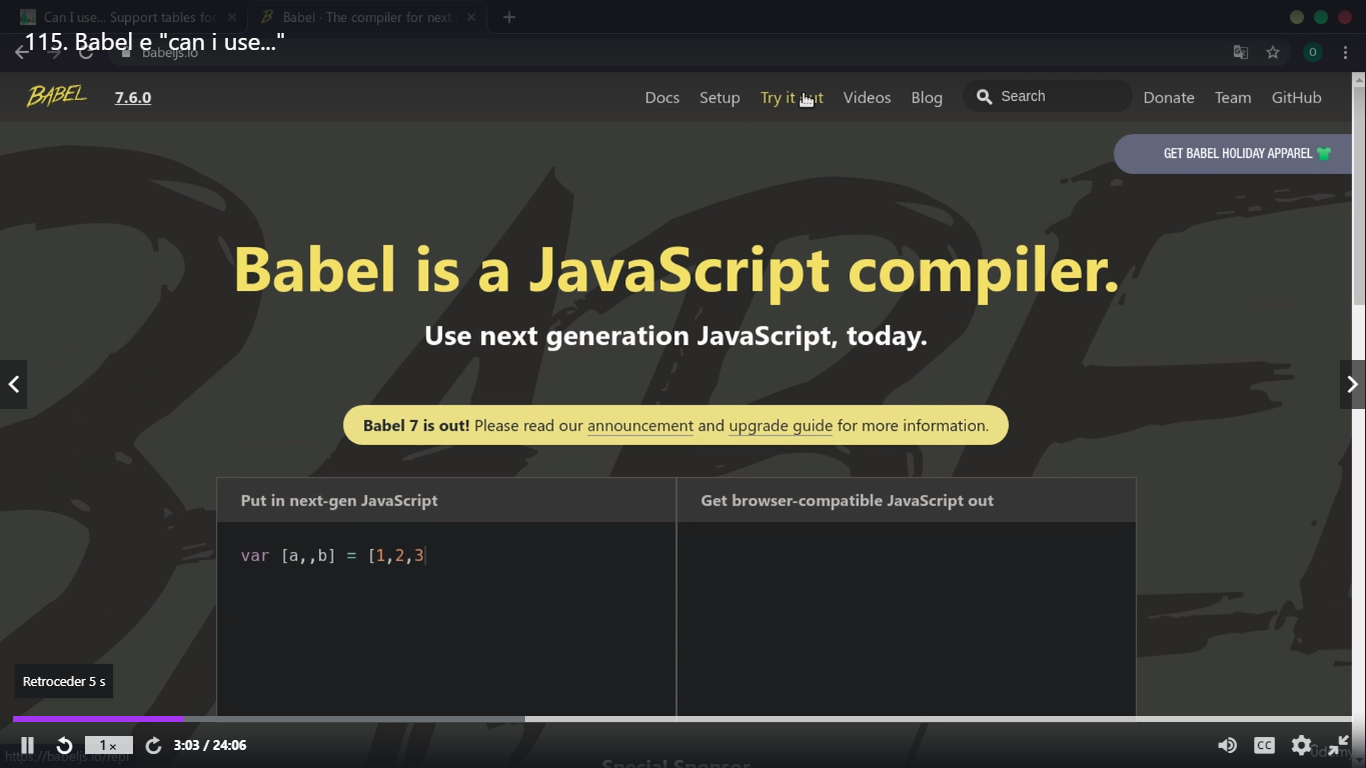
Para saber se um recurso que foi escrito em um código e suportado em algum navegador antigo, pocure pelo site can i use

https://caniuse.com



Essa e a tela do site, que ao procurar pelo recurso ‘const’, irá ver que no IE 6 ao 10 ele não suporta esse recurso, e no IE 11 ele suporta parcialmente o const, não suportando o for-in e o for-of em suas aplicações

Para resolver essa diferença de uns aceitarem e outros não os recursos que você esta programando, você pode utilizar o transpilador(tradutor e compiador do código javascript) BABEL.



No site do Babel, clique em try it out, e no campo lateral esquerdo em ENV PRESET marque Enabled, normalmente já vem marcado.

Feito isso digite seu código na caixa da esquerda que ele vai transpilar seu código para uma versão mais antiga.

Assim funciona o babel, nos iremos utilizar o webpack em nossos projetos, o webpack ultiliza o babel.

A configuração do webpack e um pouco maior

* Aula 92 Babel e “can i use”

Para iniciar o gerenciador de pacote do npm do node,entre no terminal na pasta do projeto que irá utilizar e no terminal digite:

npm init -y

ira ser criado um arquivo package.json.

esse arquivo e quem vai gerenciar quais os pacotes do node, agente precisa nesse projeto.

Em sequida precisamos instalar o babel no mesmo terminal:

npm install –save-dev @babel/cli @babel/preset-env @babel/core

com esse código sera instalado o babel no package.json la em devdependencies.

Em seguida para colocar com dev-dependecies, para caso outros desenvolvedores não queiram utilizar o babel será possível, com o código que foi dito acima já instalado.

Caso você use git, poderá criar um arquivo .gitignore, e colocar a pasta node\_modules para ele não ficar monitorando.

Agora crie seu script js normal, ao final peça para o babel converte seu script:

npx babel main.js -o bundle.js --presets=@babel/env

sera criado um novo arquivo chamado bundle.js, que irá conter o código mais antigo que será aceito em mais navegadores.

Então em vez de utilizar em um sistema real o main.js seria utilizado do bundle.js

Cada vez que for alterado o script terá que ser feito essa alteração no bundle colocando para converter esse código.

Para fazer isso de maneira altomatica basta ir no arquivo package.json, e no trecho script onde esta test, adicione uma virgula e quebre a linha, e crie um nome para sua configuração em aspas duplas sequida de dois ponto e seu comando em aspas duas.

"test": "echo \"Error: no test specified\" && exit 1",

"babel": "babel ./main.js -o ./bundle.js --presets=@babel/env -w"

Com isso cada vez que atualizar o arquivo main.js ele ira ficar recomplilando no arquivo bundle.js de maneira altomatica, mais para isso o comando deve ser executado no terminal.

npm run babel

com isso ele ficará monitorando os arquivos e caso haja alteração no arquivo, o terminal devera ser parado, ultilizando ctrl + c, para parar a execução e iniciar novamente uma outra execução do npm run babel.

* Aula 93 Webpack(boilerplate)

Inicie um novo projeto com o npm init -y

O uso do -y no final, indica que todas as perguntas em seguida serão respondida com yes.

Quando não coloca o -y, sera feitas algumas perguntas sobre o prejeto

npm i --save-dev @babel/preset-env @babel/cli @babel/core babel-loader webpack webpack-cli regenerator-runtime core-js@2

crie o arquivo .gitignore, e coloque nele o node\_modules

depois caso queira apagar essa pasta de node\_modules para fazer o envio do projeto você podepois ficara mais rápido o envio

Caso queira a pasta devolta basta da um:

npm install

que ele irar ler e ver se você tem os arquivos necessário para o projeto, caso não tenha ele ira baixar os arquivos necessário.

Agora iremos criar o arquivo de configuração do webpack

Crie um novo arquivo no seu projeto com o nome:

webpack.config.js

no arquivo webpack.config.js coloque esse código:

const path = require('path');

module.exports = {

    mode: 'development',

    entry: './src/index.js',

    output: {

        path: path.resolve(\_\_dirname, 'public', 'assets', 'js'),

        filename: 'bundle.js'

    },

    module: {

        rules: [{

            exclude: /node\_modules/,

            test: /\.js$/,

            use: {

                loader: 'babel-loader',

                options: {

                    presets: ['@babel/env']

                }

            }

        }]

    },

    devtool: 'source-map'

};

No arquivo package.json, em script coloque o código:

 "test": "echo \"Error: no test specified\" && exit 1",

 "gera": "webpack -w"

Em seguida execute o código no terminal:

Npm run gera

Crie um html na pasta public, e link o script bundle.js nesse arquivo html

Não mexa diretamente no budle.js, mexa no index.js

Agora esse será o nosso modelo de webpack

Caso venha sem node\_modules, colocque no terminal: npm install

Obs:

Exclua no package:

core-js

regenerator-runtime

exclua a pasta node\_modules e execute o código:

npm i core-js regenerator-runtime

para criar as dependencies certas

agora sim podemos utilizar esse modelo em nossos projetos

* Aula 94 ES6 modules – Import/Export

Temos dois arquivos js, index.js e modulo1.js

Esse e o modulo1.js

const nome = 'Luiz';

const sobrenome = 'Miranda';

const idade = 30;

function soma(x, y) {

    return x + y;

}

export { nome }

nesse arquivo estamos exportando a const nome e somente ela desse arquivo

agora vamos para o arquivo index.js

import { nome } from './modulo1'

console.log(nome);

aqui nesse arquivo estou pegando a const nome que está no arquivo modulo1.

Exportando mais de um const e funções

const nome = 'Luiz';

const sobrenome = 'Miranda';

const idade = 30;

function soma(x, y) {

    return x + y;

}

export { nome, sobrenome, idade, soma }

aqui estamos recebendo de modulo1, nome ,sobrenome, idade e a função soma

aqui estamos recebendo nome e colocando outro o nome dela como nome2, para não ocorrer comflito, pois nesse arquivo já tem uma const com o mesmo nome, isso gereria um erro.

import { nome as nome2, sobrenome, idade, soma } from './modulo1'

const nome = 'João';

console.log( nome, nome2, sobrenome, idade, soma(5, 5));

ou podemos alterar esse nome direto no arquivo de export:

const nome = 'Luiz';

const sobrenome = 'Miranda';

const idade = 30;

function soma(x, y) {

    return x + y;

}

export { nome as nome2, sobrenome, idade, soma }

no outro arquivo precisamos colocar como ela foi exportada:

import { nome2, sobrenome, idade, soma } from './modulo1'

console.log(nome2, sobrenome, idade);

console.log(soma(5,5));

podemos fazer a exportação direto também:

export const nome = 'Luiz';

export const sobrenome = 'Miranda';

export const idade = 30;

export function soma(x, y) {

    return x + y;

}

export class Pessoa {

    constructor(nome, sobrenome) {

        this.nome = nome;

        this.sobrenome = sobrenome;

    }

}

Aqui estamos exportando const, funções e classes

Aqui estamos recebendo esses export:

import { nome, sobrenome, idade, soma, Pessoa} from './modulo1'

console.log(nome,sobrenome, idade);

console.log(soma(5,4));

const p1 = new Pessoa('Luiz', 'Otavio');

console.log(p1);

podemos também importar tudo que poder ser exportado

import \* as meuModulo from './modulo1'

console.log(meuModulo);

console.log(meuModulo.nome,meuModulo.sobrenome,meuModulo.idade);

console.log(meuModulo.soma(5,4));

export const nome = 'Luiz';

export const sobrenome = 'Miranda';

export const idade = 30;

const cpf = 33311122;

export function soma(x, y) {

    return x + y;

}

Nesse caso a const cpf, não está sendo exportada, logo somente poderar ser usada nesse arquivo. Cpf fica sendo uma variavel privada

Agora vamos usar o default, para colocar um padrão, so modemos ter 1 default por código:

export const nome = 'Luiz';

export const sobrenome = 'Miranda';

export const idade = 30;

const cpf = 33311122;

export default function soma(x, y) {

    return x + y;

}

Somente está sendo exportada a função soma

import qualquerNome from './modulo1'

console.log(qualquerNome(5,4)); // function soma()

console.log(qualquerNome.nome); // underfined

quando coloca o import sem as chaves ele irá buscar o que o padrão daquele export

const nome = 'Luiz';

const sobrenome = 'Miranda';

const idade = 30;

const cpf = 33311122;

function soma(x, y) {

    return x + y;

}

export default (x, y) => x \* y;

export {nome, sobrenome, idade, soma}

estamos fazendo o export de uma função como sendo a default e colocando as const para serem exportada também

import padrao, {nome, sobrenome, idade, soma} from './modulo1'

console.log(padrao(5, 40));

console.log(nome, sobrenome, idade, soma(3,9));

aqui estamos pegando o import default o padrão e pegando além dele outros import