Curso de webpack

do Básico ao Avançado

O que é o webpack?

- O **webpack** é um module bundler (empacotador de módulos);
- Que recebe **entradas** (entry/entries) que são os arquivos de dependências do projeto;
- E transformar em uma **saída** (output), que condensa as dependências;
- Podemos utilizar em outros tipos de arquivos como: CSS, imagens, fontes;
- É possível também criar **configurações diferentes** para ambientes de desenvolvimento e produção;

Por que webpack?

- Reduzir a complexidade de **importação das dependências**;
- A própria importação de dependências (**CommonJS**), pois os navegadores não possuem compatibilidade com este recurso;
- **Coleta automática de dependências**, baseada no que foi importado e exportado (dependency graph);
- Possibilidade de utilizar **outros tipos de arquivos**, não só JS;
- Carrega apenas o que é utilizado, redução de carregamento de arquivos;

Webpack e SPA

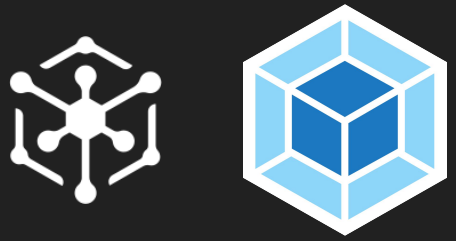
- As **SPAs** (Single Page Applications) vem dominando as arquiteturas de projetos web;
- Normalmente **é utilizado muito JS** para atingir resultados interessantes com SPAs (frameworks e libs);
- Acaba que o **webpack** ajuda a organizar todo este código JS e também otimizar a velocidade de carregamento destas páginas;

O que vamos precisar para o curso?

- Editor de texto, sugestão: **VS Code**;
- **Node.js**, para rodar os scripts de webpack;
- Alguma forma de **acessar o terminal** para rodar scripts, exemplo: terminal do Windows/Linux, git bash, VS Code;

Setup do webpack

- Para inicializar o webpack precisamos criar um projeto com o **npm init**;
- Depois vamos instalar duas dependências essenciais: **webpack e webpack-cli**;
- Depois criaremos o projeto em si com suas dependências;
- Quando o projeto estiver finalizado, utilizaremos o comando **npx webpack** para a geração do nosso bundle;



Configurações básicas

- O webpack atualmente não necessita de nenhuma configuração para funcionar;
- Porém **podemos customizá-lo** para atender alguma necessidade especificado;
- Precisamos apenas criar o arquivo **webpack.config.js** e definir nossas preferências neles;
- Ao longo do curso veremos outras opções de configuração!

Script para rodar o build

- Podemos definir um script para rodar o build do nosso webpack;
- No arquivo **package.json** vamos adicionar uma nova linha a scripts;
- O nome da chave será o comando utilizado em **npm run <comando>;**
- E o valor será **webpack**, para realizar o build;

Introdução ao webpack

Conclusão da seção

Conceitos fundamentais

Introdução da seção

Entrada e saída

- Estes são os conceitos mais importantes do webpack, que determinam seu funcionamento;
- **Entry** ou entry point é o ponto inicial, onde as dependências do projeto são declaradas para o webpack, geralmente o arquivo **index.js**;
- **Output** é a saída do empacotamento do webpack, por default fica na pasta **dist** e o arquivo é o **main.js**;
- Todas estes padrões podem ser alterados em **webpack.config.js**;

Loaders

- **Loader** é o recurso que permite o webpack processar diversos tipos de arquivos, além de JavaScript;
- A cada tipo de arquivo novo que vai ser processado no projeto, **precisamos configurar no webpack.config.js**;
- Duas propriedades precisam ser definidas: **test**, para o tipo de arquivo, e **use**, para o loader a ser utilizado;
- Loaders precisam ser instalados!

Plugins

- **Plugins** são funcionalidades que podem ser adicionadas ao nosso projetos;
- Como por exemplo: minificar o JS para deixar o carregamento mais rápido;
- Precisamos instalar e inicializar cada um dos plugins, eles devem ser adicionados no arquivo **webpack.config.js**;

Mode

- **Mode** ou modo é a forma que o webpack vai rodar no projeto que está sendo executado;
- Podemos separar como development, production ou none;
- Criando **configurações isoladas** para cada um deles;
- Definimos o modo do projeto também no arquivo **webpack.config.js**;

Compatibilidade

- O webpack roda em cima do **Node.js 10.13+**;
- O navegador precisa suportar pelo menos todos os **recursos de ES5**, versão do JavaScript;
- Há maneiras de rodar o webpack em navegadores mais antigos, utilizando **polyfills**;

Conceitos fundamentais

Conclusão da seção

Loaders

Introdução da seção

Importando CSS

- Para importar CSS com o webpack vamos precisar de **loaders**;
- Neste caso será **style-loader e css-loader**;
- Obs: quando há mais de um loader, a **ordem importa**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Depois deste processo podemos importar arquivos de CSS;

Importando imagens

- Para importar imagens com o webpack vamos precisar de um **loader**;
- Neste caso será o **file-loader**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Depois deste processo podemos importar a imagem no projeto;

Importando SASS

- Para importar SASS com o webpack vamos precisar de **loaders**;
- Neste caso será **style-loader, css-loader e sass-loader**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Depois deste processo podemos importar arquivos de SASS;

Importando funcionalidades ES6+

- Para utilizar funcionalidades do ES6+ precisamos utilizar um **loader**;
- Neste caso será o **babel-loader**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Dependendo da funcionalidade, precisamos adicionar também plugins;

Importando HTML

- Para importar HTML com o webpack vamos precisar de um **loader**;
- Neste caso será **html-loader**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Depois deste processo podemos importar arquivos de HTML no projeto;

Importando arquivos de texto

- Para importar arquivos de texto com o webpack vamos precisar de um **loader**;
- Neste caso será **raw-loader**;
- O loader precisa ser adicionado no arquivo de configuração;
- Precisamos também **instalar como dependência de desenvolvimento**;
- Depois deste processo podemos ler o conteúdo de arquivos de texto no projeto;

Importando JSON

- Após a versão 2 do webpack o **JSON se tornou um tipo de dado padrão para o webpack**;
- Ou seja, **não precisamos de loaders**;
- Podemos simplesmente dar o require no arquivo e introduzir a uma variável;
- Depois podemos utilizar o seu conteúdo;

Loaders

Conclusão da seção

Plugins

Introdução da seção

Extração de CSS em arquivo

- Para extrair CSS em um arquivo separado utilizamos um **plugin**;
- O nome desse plugin é **MiniCSSExtractPlugin**;
- Precisamos adicioná-lo ao arquivo de configurações **webpack.config.js**;
- Este plugin vai entrar no lugar de style-loader, para realizar as modificações necessárias e gerar um arquivo;

Minificar JavaScript

- Para minificar JS com o webpack podemos utilizar um **plugin**;
- O nome desse plugin é **TerserPlugin**;
- **Obs:** em modo de produção, não é necessário;
- Precisamos adicioná-lo ao arquivo de configurações **webpack.config.js**;
- Este plugin vai gerar um JS minificado para melhorar a performance;

Criando constantes globais

- Para definir constantes acessíveis em todas a nossa aplicação **plugin**;
- O nome desse plugin é **DefinePlugin**;
- Precisamos adicioná-lo ao arquivo de configurações **webpack.config.js**;
- Este plugin vai disponibilizar constantes em todo o nosso software, podemos utilizar para chaves de API, por exemplo;

Geração de arquivo HTML

- Para gerar um arquivo HTML baseado em nossas configs usamos um **plugin**;
- O nome desse plugin é **HtmlWebpackPlugin**;
- Precisamos adicioná-lo ao arquivo de configurações **webpack.config.js**;
- Este plugin vai gerar um arquivo de HTML em todas as builds, linkando automaticamente os arquivos que precisamos para o projeto funcionar;

Resolvendo problemas de cache

- Podemos eliminar o problema de cache de arquivos estáticos adicionando um **nome diferente para cada build**;
- No **filmename** precisamos substituir o nome para **[contenthash]**;
- A cada build um arquivo com nome diferente será criado;
- Obrigando o navegador a baixá-lo novamente;

Limpando arquivos desnecessários

- Podemos acabar com o problema de arquivos desnecessários com um **plugin**;
- O nome dele é **CleanWebpackPlugin**;
- A cada build vai apagar os arquivos que não são mais necessários e depois gerar os que são normalmente;

Plugins

Conclusão da seção

Mode

Introdução da seção

Modo de desenvolvimento

- O modo **development** é o modo indicado para quando estamos desenvolvendo aplicações;
- Alteramos o mode em webpack.config.js;
- Este modo **não é otimizado**, por exemplo: não minifica os arquivos;
- Após alterar o modo precisamos gerar uma build para funcionar;

Modo de produção

- O modo **production** é o modo indicado para quando fazemos o deploy da aplicação;
- Alteramos o mode em webpack.config.js;
- Este modo **é otimizado**, utilizando plugins para melhorar a performance, por exemplo;
- Após alterar o modo precisamos gerar uma build para funcionar;

Separando dev e prod

- Podemos separar ambientes criando **dois arquivos de configuração**;
- Depois vamos criar **dois scripts de npm**, um para cada ambiente;
- Então podemos decidir na hora da build qual ambiente estamos, criando os arquivos corretos para cada situação;
- Os arquivos do projeto serão gerados normalmente como se tivesse apenas um;

Webpack dev server

- Podemos criar um servidor com um pacote do webpack;
- Desta maneira **se torna mais fácil desenvolver com o webpack**;
- Vamos precisar **configurar o servidor** no arquivo de config;
- Podemos também adicionar o script de rodar o servidor com a build do ambiente de desenvolvimento;

Mode

Conclusão da seção

Integrações

Introdução da seção

Integrando o jQuery

- Primeiramente vamos instalar no projeto pelo npm o **jQuery**;
- Realizar a importação do mesmo onde for necessário;
- Rodar a build do projeto;
- E por fim inicializar o servidor do web / abrir arquivo no navegador;

Integrando o Google Fonts

- Primeiramente vamos fazer o **download da fonte do projeto**;
- Podemos **converter o arquivo ttf em mais formatos** para aumentar a compatibilidade entre navegadores;
- Criar uma pasta e inserir as fontes nela;
- Inserir a fonte com a regra de **@font-face** do CSS;
- **Importar o CSS** na página e **configurar o file-loader**;
- Buildar o projeto e rodar o servidor;

Integrando o Font Awesome

- Primeiramente instalar os **pacotes referentes ao Font Awesome** com npm;
- Importar os pacotes **library e dom** de fontawesome-svg-core;
- Importar o ícone de **free-solid-svg-icons**;
- Inserir o ícone com **library.add**;
- E por fim utilizar o método **dom.watch** para substituição;
- Build e depois rodar o servidor;

Integrando o Bootstrap

- Instalar o **Bootstrap** com o npm, e também as suas dependências: **jQuery e popper**;
- Importar o JS e o CSS do Bootstrap no projeto;
- Rodar o build do webpack;
- E por fim inicializar o servidor ou abrir o HTML no navegador;

Integrando o Bulma

- Primeiramente vamos instalar algumas dependências como o **framework e os loaders** necessários;
- Definir as **regras de SASS** no arquivo de config;
- **Criar um arquivo index.js** com a importação de um SASS;
- No arquivo de SASS adicionar importações do Bulma;
- Criar um arquivo de HTML com os componentes;
- Buildar e rodar o servidor;

Integrando o React

- Primeiramente vamos instalar os pacotes **react e react-dom**;
- Depois vamos precisar do **babel e alguns presets**;
- Vamos criar um arquivo **.babelrc** para inserir os presets, deixando mais organizado;
- Vamos utilizar o webpack server e também o html-webpack-plugin;
- Em **src** vamos adicionar um index.js e também um componente React;
- Por fim configurar, buildar e rodar;

Integrando o Vue

- Primeiramente instalaremos as dependências: **vue, vue-router e core-js**;
- Em seguida os **pacotes de babel e o webpack**;
- Inserir os presets no **.babelrc**;
- Vamos criar a pasta **public e src** com os arquivos base da aplicação;
- Podemos criar um script para rodar o servidor do webpack;
- Rodar a build e ligar o servidor;

Integrando o Express

- Primeiramente vamos instalar o **webpack e o express**;
- Depois precisamos configurar em src o arquivo que vai ligar o servidor de express;
- Criar o **script de inicialização**;
- Rodar o build e iniciar o servidor;

Integrando o TypeScript

- Primeiramente vamos instalar o pacote **typescript** e o loader **ts-loader**;
- Depois criamos o arquivo de configurações **tsconfig.json**;
- Podemos também inserir um arquivos de TypeScript: **index.ts**;
- Modificamos para o **entry** para ser o index.ts e configuramos o loader;
- Para finalizar, geramos o build;

Integrações

Conclusão da seção

Recursos do Webpack

Introdução da seção

Shimming

- É uma forma de **não utilizar variáveis globais de módulos** no código;
- Exemplo: $ do jQuery;
- Deixando o webpack se encarregar de invocar o módulo quando o encontrar;
- Precisamos utilizar o **ProviderPlugin** para realizar esta ação;
- E utilizar este recurso é sugerido como uma boa prática pelo Webpack;

Melhorar performance do build

- **Inserir um path para limitar as pastas dos loaders** tem de serem executados;
- **Utilizar o mínimo de plugins** possível, e não utilizar plugins de produção (Terser);
- **Utilizar o servidor do Wepback**, pois compila na memória, não no disco;
- Criar um **chunk para o runtime**, na configuração de optimization, definir runtimeChunk como true;

HMR

- O HMR (**Hot Module Replacement**) é uma funcionalidade que permite recarregar algumas funcionalidades sem buildar novamente;
- Pertence ao webpack server;
- Precisamos adicionar a opção de **hot** como **true**;

Code Splitting

- Podemos dividir os arquivos de código que geramos pelo webpack, isso é chamado de **code splitting**;
- Em **entry** vamos precisar definir os arquivos que serão separados;
- E precisamos também deixar o filename de **output** dinâmico, com **[name]**;
- E então teremos o projeto rodando normalmente, porém com o código dividido

Prevenção de duplicação

- Se importamos uma lib em dois arquivos de code splitting, **podemos gerar uma duplicidade**;
- Para resolver isso criamos uma regra de **dependOn**;
- A chave dependOn fica em **entry**, em cada declaração;
- Esta regra denomina uma biblioteca em comum que será configurada no nome que colocamos como o pacote duplicado;
- A biblioteca compartilhada terá um **arquivo gerado apenas para ela**;

Análise do bundle

- Podemos checar como está o nosso projeto no seguinte site: https://alexkuz.github.io/webpack-chart/
- Vamos precisar **gerar um arquivo json** de nossas dependências;
- O comando é: **npx webpack --profile --json=stats.json**
- Desta maneira podemos analisar o que está sendo importado no nosso projeto;

Lazy Load

- Podemos carregar um componente apenas quando ele precisar ser utilizado;
- Desta maneira podemos **otimizar o carregamento da página**;
- Para realizar esta ação podemos, por exemplo, **atrelar a importação a evento**;
- Podemos ver o carregamento do arquivo na **aba Network**;

Recursos do Webpack

Conclusão da seção

Module Federation

Introdução da seção

O que é Module Federation?

- Uma funcionalidade do Webpack para criar **múltiplas aplicações em um projeto**;
- A ideia é que elas **não** tenham dependências compartilhadas entre si;
- Podemos simular a **arquitetura de micro-frontend** com este recurso;
- Conseguiremos desenvolver várias aplicações distintas que são coordenadas por uma build;