**Configurando o Ambiente**

**Criando a estrutura do projeto**

crie uma pasta chamada **src** dentro dela crie o arquivo **index.jsx**

import React from 'react';

function App(){

return <h1>Helloo World</h1>

}

Crie uma pasta chamada **public** crie o arquivo **index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Projeto</title>

</head>

<body>

<div id="root"></div>

<script src="../dist/bundle.js"></script>

</body>

</html>

*// usamos esse i****d****="root" para mostrar todos os componentes na tela dentro dessa div, padrão do react*

**Criar arquivo json**

As orientações abaixo devem ser feitas no terminal basta estar na pasta do projeto

Ex: C:\Users\ADMIN\Desktop\Projeto> **"comando no terminal é escrito aqui "**

para criar um arquivo json mostrando as dependencias e versões use o comando

**yarn init -y**

adicione o react

**yarn add react**

adicionada abiblioteca react com os modulos e dependencias react

**yarn react-dom**

A biblioteca react dom permite que react se comunique com a arvore de elementos html para que ele saiba como criar elementos em tela e deletar etc "

**Configurando Babel**

adicionando e configurando babel:

**yarn add @babel/core @babel/cli @babel/preset-env -D**

babel ferramenta que recodifica codigo para que todo navegador entenda.

criar arquivo nomeado como **babel.config.js**

digite o codigo a seguir dentro:

**module.exports={**

**presets:['@babel/preset-env',**

**['@babel/preset-react',{**

**runtime:'automatic'**

**}]**

**]**

**};**

comando de execução babel:

**yarn babel src/index.js --out-file dist/bundle.js**

esse comando acima escolhe o arquivo a ser recodificado e o diretorio a ser criado para novo codigo

o babel não entende html ou jsx , então inserimos novo recurso.

**yarn add @babel/preset-react -D**

agora o balbel consegue recodificar jsx para diversos navegadores entenderem.

o babel esta configurado.

**Configurando webpack**

webpack permite configurar e ensinar a aplicação tratar cada tipo de arquivo,convertendo a tipos o que navegador entenda.



instalar e configurar:

**yarn add webpack webpack-cli -D**

Crie o arquivo de configuração **webpack.config.js**

digite o codigo:

**const path = require ('path')**

**module.exports={**

**mode: 'development',**

**entry: path.resolve(\_\_dirname,'src', 'index.jsx'), //local arquivo principal do app**

**output: {**

**path: path.resolve(\_\_dirname, 'dist'), //local de destino do codigo**

**filename: 'bundle.js'**

**},**

**resolve : {**

**extensions:['.js','.jsx'], // permite entender as duas extenções**

**},**

**module:{ //configura comportamento quando importa cada tipo de arquivo**

**rules:[ //array de regras**

**{**

**test:/\.jsx$/, //expressão regular que verifica se é um arquivo js ou não**

**exclude:/node\_modules/, //exclui todos arquivos que estão na pasta**

**use:'babel-loader' // integração entre babel e webpack**

**}]**

**}**

**};**

terminal:

**yarn add babel-loader -D**

**yarn webpack**

**Servindo Html Estático**

Para injetar js em html automaticamente.

**yarn add html-webpack-plugin -D**

Permite injetar o bundle no arquivo html

acrescentar no webpack.config.js

**const HtmlWebpackPlugin = require('html-webpack-plugin')**

**plugins:[**

**new HtmlWebpackPlugin({**

**template: path.resolve(\_\_dirname,'public','index.html')**

**})**

**],**

O arquivo esta atualmente assim.

**const path = require ('path')**

**const HtmlWebpackPlugin = require('html-webpack-plugin')**

**module.exports={**

**mode: 'development',**

**entry: path.resolve(\_\_dirname,'src', 'index.jsx'),**

**output: {**

**path: path.resolve(\_\_dirname, 'dist'),**

**filename: 'bundle.js'**

**},**

**resolve : {**

**extensions:['.js','.jsx'],**

**},**

**plugins:[**

**new HtmlWebpackPlugin({**

**template: path.resolve(\_\_dirname,'public','index.html')**

**})**

**],**

**module:{**

**rules:[**

**{**

**test:/\.jsx$/,**

**exclude:/node\_modules/,**

**use:'babel-loader'**

**}]**

**}**

**};**

**Webpack Dev Server**

**yarn add webpack-dev-server -D**

Permite que webpack esxute as alterações e ja faça as modificações sem que nos precisemos ficar usando **yarn webpack**

Acrescentar a config no webpackconfig

Mais uma propriedade :

**devServer:{**

**contentBase: path.resolve(\_\_dirname,'public')**

**},**

basta executar o comando

**yarn webpack serve**

agora toda alteração esta sendo atualizada automaticamente e fica disponivel no endereço <http://localhost:8080/>

**webpack.config.js**

**const path = require ('path')**

**const HtmlWebpackPlugin = require('html-webpack-plugin')**

**module.exports={**

**mode: 'development',**

**entry: path.resolve(\_\_dirname,'src', 'index.jsx'),**

**output: {**

**path: path.resolve(\_\_dirname, 'dist'),**

**filename: 'bundle.js'**

**},**

**resolve : {**

**extensions:['.js','.jsx'],**

**},**

**devServer:{**

**contentBase: path.resolve(\_\_dirname,'public')**

**},**

**plugins:[**

**new HtmlWebpackPlugin({**

**template: path.resolve(\_\_dirname,'public','index.html')**

**})**

**],**

**module:{**

**rules:[**

**{**

**test:/\.jsx$/,**

**exclude:/node\_modules/,**

**use:'babel-loader'**

**}]**

**}**

**};**

**Ultilizando source maps**

Agora sera configurado source maps para ajudar a mostrar o erro de forma correta.

devtool é uma configuração podendo conter opçãp 'eval-source-map

**Ambiente Dev e Produção**

Essa configurações a seguir são para definir ambiente de dev ou produção, um padrão adotado nesse conteudo.

**const isDevelopement = process.env.NODE\_ENV != 'production';**

**module.exports={**

**mode: isDevelopement ? 'development': 'production',**

**devtool: isDevelopement ? 'eval-source-map' : 'source-map',**

O arquivo webpack.config.js deve estar do seguinte modo

**const path = require ('path')**

**const HtmlWebpackPlugin = require('html-webpack-plugin')**

**const isDevelopement = process.env.NODE\_ENV != 'production';**

**module.exports={**

**mode: isDevelopement ? 'development': 'production',**

**devtool: isDevelopement ? 'eval-source-map' : 'source-map',**

**entry: path.resolve(\_\_dirname,'src', 'index.jsx'),**

**output: {**

**path: path.resolve(\_\_dirname, 'dist'),**

**filename: 'bundle.js'**

**},**

**resolve : {**

**extensions:['.js','.jsx'],**

**},**

**devServer:{**

**contentBase: path.resolve(\_\_dirname,'public')**

**},**

**plugins:[**

**new HtmlWebpackPlugin({**

**template: path.resolve(\_\_dirname,'public','index.html')**

**})**

**],**

**module:{**

**rules:[**

**{**

**test:/\.jsx$/,**

**exclude:/node\_modules/,**

**use:'babel-loader'**

**},]**

**}**

**};**

Configurando variáveis de ambiente.

**yarn add cross-env -D**

No packagejson vamos configurar os comandos scripts.

**"scripts" : {**

**"dev": "webpack serve",**

**"build" : "cross-env NODE\_ENV= production webpack"**

**},**

**Importando arquivos Css**

Agora temos que configurar o projeto para entender arquivos css,

então no webpackconfig criaremos uma nova regra em rules.

**yarn add style-loader css-loader -D**

**rules:[**

**{**

**test:/\.jsx$/,**

**exclude:/node\_modules/,**

**use:'babel-loader'**

**},**

**{ //essa é a nova regra**

**test:/\.css$/,**

**exclude:/node\_modules/,**

**use:['style-loader' ,'css-loarder'],**

**}**

**]**

sendo assim podemos importar arquivos css serão entendidos e recodificados.

**Ultilizando SASS**

agora configuraremos o Sass um pré-processador de css conseguindo ter funcionabilidades a mais no css.

**yarn add node-sass -D**

atualize o arquivo webpack.config.js

subistitua essa configuração pela existente

**{**

**test:/\.scss$/,**

**exclude:/node\_modules/,**

**use:['style-loader' ,'css-loader','sass-loader'],**

**}**

Webpack.Config.js ficara assim :



imagem da estrutura do projeto:

