Node notas

npm init -y inicializa tu proyecto (crea fichero package.json)

yarn init -y

npm install package --save (instalar libreria)

yarn add package

npm install package --save-dev instala una librería como dependencia de desarrollo

yarn add package --dev

comandos buenos de webpack (se instala webpack y el CLI command line interface)

Añadir webpack y el cli

yarn add webpack webpack-cli --dev

Añadir babel para transpilar (CLI, core es el nucleo, preset-env son los settings para transpilar mas comun )

yarn add @babel/cli @babel/core @babel/preset-env --dev

Añadir loaders para unir babel a webpack

yarn add babel-loader --dev

Añadir loaders para utilizar sass

Yarn add sass-loader –dev

Para usarlo

  {

        test: /\.scss$/,

        exclude: /node\_modules/,

        use: [

          "style-loader",

          {

            loader: "css-loader",

            options: {

              modules: {

                exportLocalsConvention: "camelCase",

                localIdentName: "[path][name]\_\_[local]--[hash:base64:5]",

                localIdentContext: path.resolve(\_\_dirname, "src"),

              },

            },

          },

          "sass-loader",

        ],

Loaders para trabajar con imagenes (está incluido en webpack ahora ya está incluido )

 {

        test: /\.(png|jpg|gif)$/,

        type: "asset/resource",

      },

Html loader para renderizar bien el html si ejecutamos servidores locales, como lite-server o go-live(VsCode)

Yarn add html-loader –dev

Para usarlo añadir dentro de module.rules

 {

        test: /\.html$/,

        loader: "html-loader",

      },

Crear servidor de transpilacion en memoria

yarn add webpack-dev-server --dev

Plugin para manejar html

yarn add html-webpack-plugin –dev

lo importamos

const HtmlWebpackPlugin = require("html-webpack-plugin");

luego añadirlo al fichero webpack.config en la sección de plugins

 plugins: [

    new HtmlWebpackPlugin({

      filename: "index.html", // Fichero en dist

      template: "./index.html", // Fichero de entrada ./src

      scriptLoading: "blocking",

    }),

  ],

Plugin para sacar los css como ficheros css

yarn add mini-css-extract-plugin --dev

para usarlo luego

  test: /\.scss$/,

        exclude: /node\_modules/,

        use: [

          MiniCssExtractPlugin.loader,

          {

            loader: "css-loader",

            options: {

              modules: {

                 exportLocalsConvention: "camelCase",

                localIdentName: "[path][name]\_\_[local]--[hash:base64:5]"

              },

            },

          },

          "sass-loader",

        ],

La parte en verde es la configuración que se usa cuando queremos trabajar con múltiples ficheros css

Export locals convention es basicamente para convertir los ficheros que tienen – a camelCase

localIdenName: nombre de la clase que aparece luego

Luego para utilizar los ficheros si queremos evitarnos colisión de estiles hacemos algo como esto

import classes from "./averageComponentStyles.scss";

y lo usas como

  <p className={classes.resultBackground}></p>

Plugin para añadir variables de entorno .env

yarn add dotenv-webpack –dev

Text

Description automatically generatedse genera el fichero de variables de entorno

para usarlo se añade a webpack

const dotEnv = require("dotenv-webpack");

y luego lo usas en los plugins

 plugins: [

    new MiniCssExtractPlugin({

      filename: "[name].[contenthash].css",

    }),

    new dotEnv({

      path: "./prod.env",

    }),

Para ejecutar el proyecto se puede usar el

Yarn add lite-server -g (para intstalarlo en global)

Luego para ejecutarlo es solo poner en el terminal

Lite-server

Para ir hacienda control de errores, chequeo de tipos (para typescript) todo el tiempo se usa este paquete

Yarn add npm-run-all --dev

Para usarlo luego en el package.json se debe añadir

Run-p seria para ejecutar todo en npm-run all

El -l es para marcar con colores los logs

 "scripts": {

    "start": "run-p -l type-check:watch start:dev",

    "build:prod": "webpack --config webpack.prod.js",

    "build:dev": "webpack --mode development --config webpack.dev.js",

    "type-check": "tsc --noEmit",

    "type-check:watch": "yarn type-check --watch",

    "start:dev": "webpack serve --mode development --config webpack.dev.js",

    "start:prod": "webpack serve --config webpack.prod.js",

    "build:perf": "yarn type-check && webpack --config webpack.perf.js"

  },

Extras de webpack

Se debe crear un fichero webpack.config.js para otras configuraciones relacionadas con webpack. Importante los loaders se utilizan directamente pero los plugins se importan

Para que la aplicación vaya siempre al contexto del src correcto se usa la línea path que es un método de node

const path = require("path");

y en el module.export declaro el contexto de trabajo

module.exports = {

  context: path.resolve(\_\_dirname, "./src"),

  resolve: {

Si instalas react y typescript pero tienes también ficheros ts debes poner en el module.rules

Test: /\.tsx?$/ (indicando que aceptas ts y tsx)

 module: {

    rules: [

      {

        test: /\.tsx?$/,

        exclude: /node\_modules/,

        loader: "babel-loader",

y luego en el module.exports.resolve le dices que extensiones buscara

module.exports = {

  context: path.resolve(\_\_dirname, "./src"),

  resolve: {

    extensions: [".js", ".ts", ".tsx"],

  },

Si solo queremos ver los errores en los logs

devServer: {

    port: 8081,

    devMiddleware: {

      stats: "errors-only",

    },

  },

Webpack merge para combinar webpacks configs

Yarn add webpack-merge –dev

Para usarlo

const common = require("./webpack.common.js"); // de este vas a heredar

const { merge } = require("webpack-merge")

module.exports = merge(common, { //aqui todo lo que quieras poner nuevo

  module: {

    rules: [

luego en el package.json debes tirar de uno u otro fichero de webpack

 "build:dev": "webpack --mode development --config webpack.dev.js",

"start:dev": "webpack serve --mode development --config webpack.dev.js",

En webpack para depurar código en el navegador se puede usar el eval-source-map (solo para testing)

 devtool: "eval-source-map", //para en prod poder mirar codigo en el source webpack

para hacer análisis se puede usar el webpack analyzer

yarn add webpack-bundle-analyzer –dev

para usarlo se genera un fichero webpack.perf.js (el perf es inventado) con una configuración parecida a esta (en este caso se quiere hacer el análisis en en el webpack de producción)

const { merge } = require("webpack-merge");

const prod = require("./webpack.prod.js");

const { BundleAnalyzerPlugin } = require("webpack-bundle-analyzer");

module.exports = merge(prod, {

  plugins: [new BundleAnalyzerPlugin()],

});

Extras de babel

También generamos un .babelrc, este fichero igualmente esta relacionado con las configuraciones de babel. Babel es como el transpilador, y debe conocer los settings de lo que se quiere interpretar, en este caso preset-env (los presets standars de babel), preset-react (para que interprete que el código es de react) y lo mismo con preset-typescript

{

  "presets": ["@babel/preset-env", "@babel/preset-react", "@babel/preset-typescript"]

}

Para usar parcel

Yarn add parcel –dev