**Webpack Config**

**Webpack** : [Webpack](https://webpack.js.org/) is a *static module bundler* for JavaScript applications. It takes [modules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules), whether that’s a custom file that we created or something that was installed through NPM, and converts these modules to static assets.

When we had very few JS scripts on a webpage, there were two ways to load JavaScript.

<html> <body> <script src="index.js"></script> </body></html>

and

<html> <body> <script> var foo = "bar"; console.log(foo); </script> </body></html>

This method does not scale, and when we have many scripts, it becomes a problem to load these scripts because of network bottleneck. If we keep them all in one file, then it becomes unmaintainable and causes problems with the name and scope of variables.

### Using a configuration with webpack

Webpack doesn’t require any configuration, but most projects will need a more complex setup, which is why webpack supports a configuration file. Add a file with the name webpack.config.js in the dummy-project directory.

Example ****webpack.config.js****

module.exports = {

    entry: './src/index.ts',

    devtool: 'inline-source-map',

    module: {

        rules: [

            {

                test: /\.tsx?$/,

                use: 'ts-loader',

                exclude: /node\_modules/,

            },

        ],

    },

    resolve: {

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

    },

    output: {

        filename: 'bundle.js',

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

    },

    target: 'node',

    node: {

        // Need this when working with express, otherwise the build fails

        \_\_dirname: false,   // if you don't put this is, \_\_dirname

        \_\_filename: false,  // and \_\_filename return blank or /

    },

    externals: [nodeExternals()], // Need this to avoid error when working with Express

};

The following are some configurable concepts of webpack.

* ****Entry****. *Entry* defines the entry-point for the application. It is the first module that webpack will process to build its dependency graph.

module.exports = { entry: './path/to/my/entry/file.js'};

* ****Output****. The output property tells webpack where to store the bundles it creates and how to name these files. The default location is ./dist/main.js for Javascript files and also stores other generated files in the ./dist folder.

module.exports = { output: { filename: 'my-first-bundle.js', pathname: \_\_dirname + '/dist' }}

* ****Loaders****. Out of the box, webpack only understands JavaScript and JSON files. Loaders allow webpack to process other types of files and convert them into valid modules that can be consumed by your application and added to the dependency graph.

moduel.exports = { module: { rules: [ {test: /\.txt$/, use: 'raw-loader'} ] }}

The test property identifies which file should be transformed. The use property indicates which loader should be used.

* ****Plugins****. While loaders are used to transform certain types of modules, plugins can be leveraged to perform a wider range of tasks like bundle optimization, asset management and injection of environment variables. In order to use a plugin, you need to require() it and add it to the plugins array. Most plugins are customizable through options.

const HtmlWebpackPlugin = require('html-webpack-plugin');module.exports = { plugins: [ new HtmlWebpackPlugin({template: './src/index.html'}) ]};

* ****Mode****. The mode can be set to production or development with their usual meanings.

We can add webpack to npm scripts in package.json so as to ease our development process. Add "build": "webpack", to package.json and run the command npm run build.