Hands-on Lab: Build and Deploy a Website Using Webpack

Skills Network

Estimated time needed: 30 minutes

In this lab, you will learn how to package your web application using Webpack.

The lab is a continuation of the Fruit Marketplace.

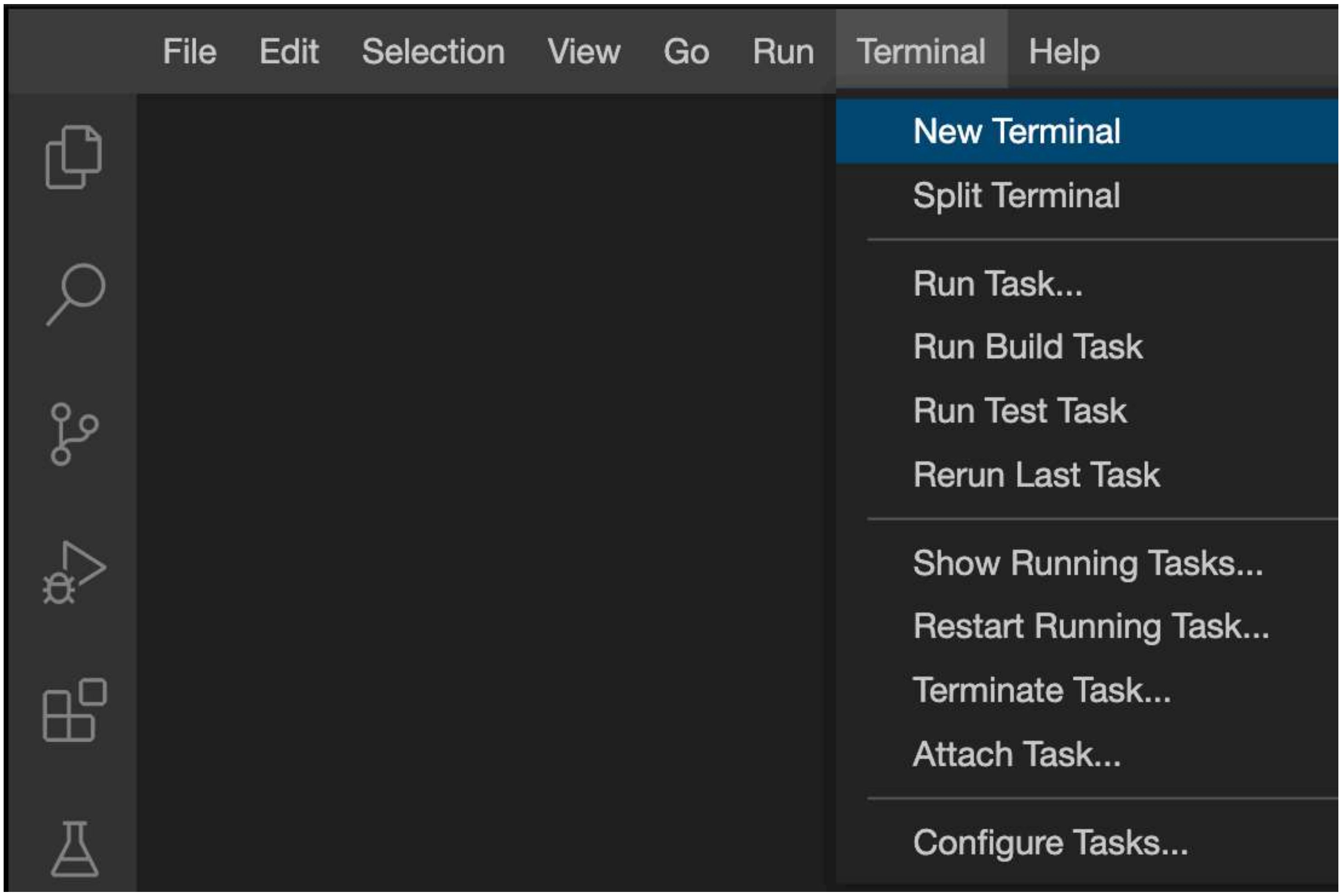
Objectives:

After completing this lab, you will be able to:

- Use Webpack to package images and CSS files.

Set-up : Create application

1. Open a terminal window by using the menu in the editor: **Terminal** > **New Terminal**.



2. If you are not currently in the project folder, copy and paste the following code to change to your project folder.

```
cd /home/project
```

Copied! Executed!

3. Run the following command to clone the Git repository that contains the starter code needed for this lab if the Git repository doesn't already exist.

```
[ ! -d 'ghwey-build-deploy-webpack' ] && git clone https://github.com/ibm-developer-skills-network/ghwey-build-deploy-webpack.git
```

Copied! Executed!

4. Change to the directory **ghwey-build-deploy-webpack** to start working on the lab.

```
1. 1
1. cd ghwey-build-deploy-webpack
```

Copied! Executed!

5. List the contents of this directory to see the artifacts for this lab.

```
1. 1
1. ls
```

Copied! Executed!

6. Run the following command on the terminal to host your web page.

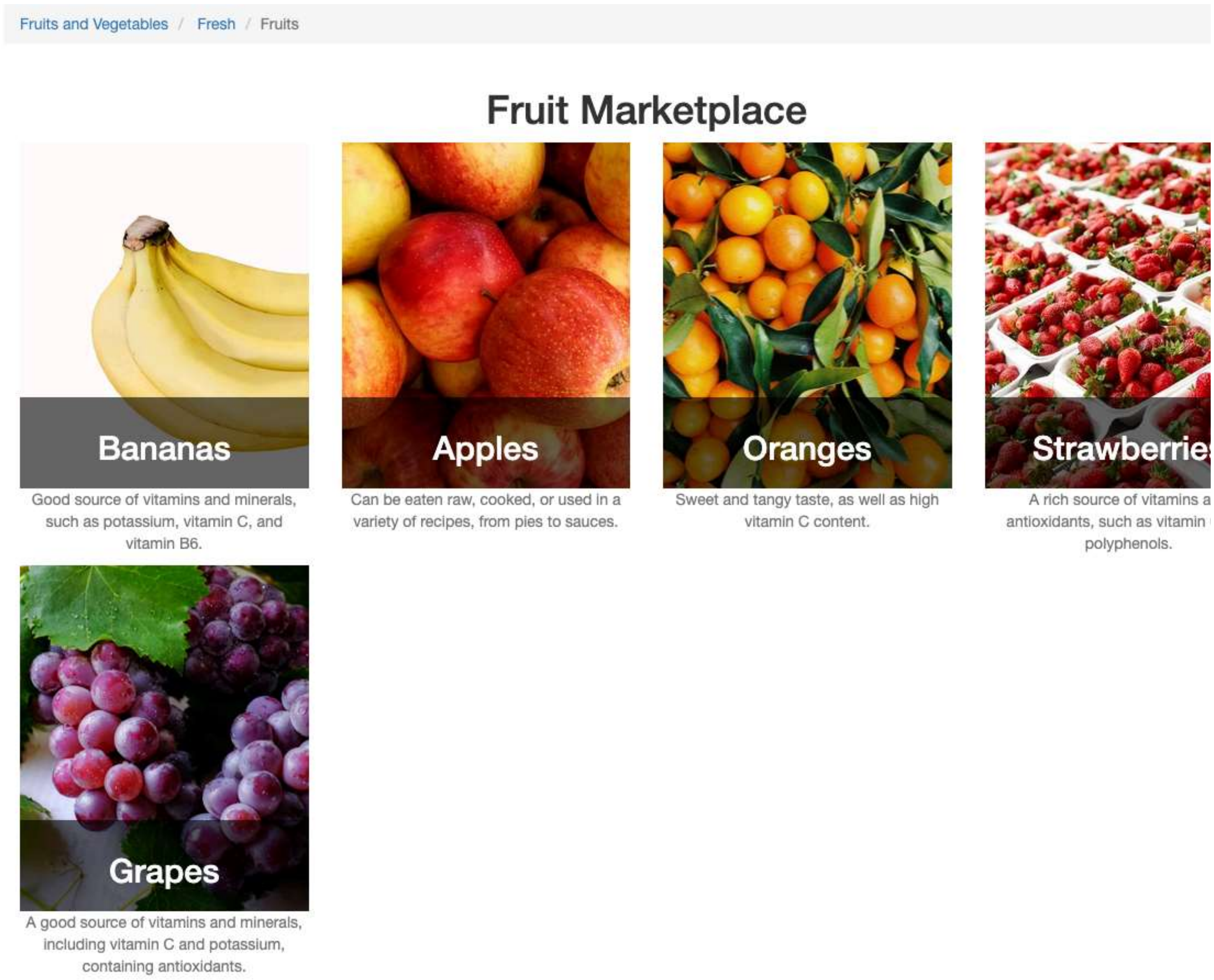
```
1. 1
1. python3 -m http.server
```

Copied! Executed!

7. To test your application in your browser, run the application first.

Launch Application

8. It will look like this:

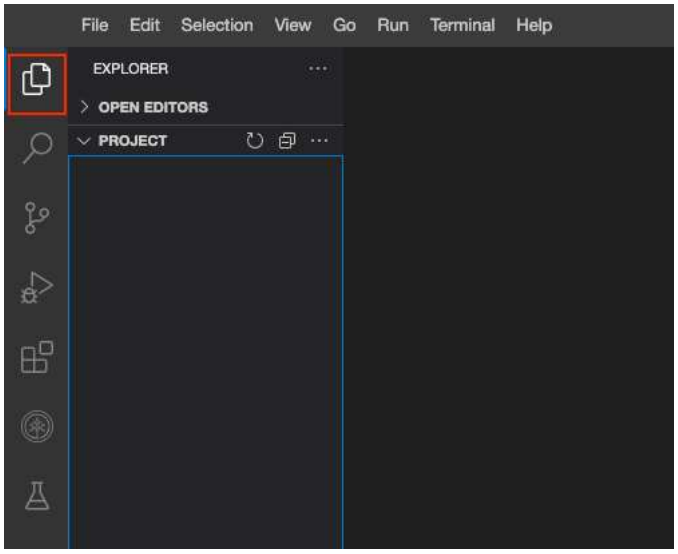

Fruits and Vegetables / Fresh / Fruits
Fruit Marketplace
Bananas
Good source of vitamins and minerals, such as potassium, vitamin C, and vitamin B6.
Apples
Can be eaten raw, cooked, or used in a variety of recipes, from pies to sauces.
Oranges
Sweet and tangy taste, as well as high vitamin C content.
Strawberrie
A rich source of vitamins a antioxidants, such as vitamin polyphenols.
Grapes
A good source of vitamins and minerals, including vitamin C and potassium, containing antioxidants.

9. You can copy the URL and run it in your own browser.

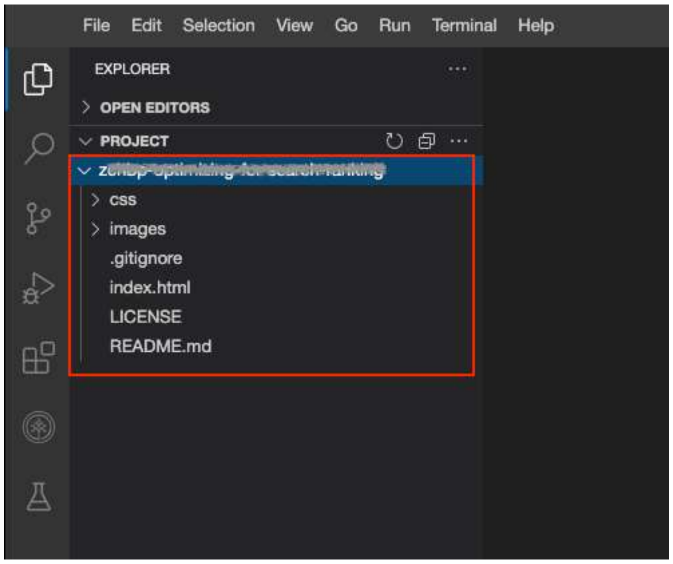
Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your projet, in Cloud IDE.

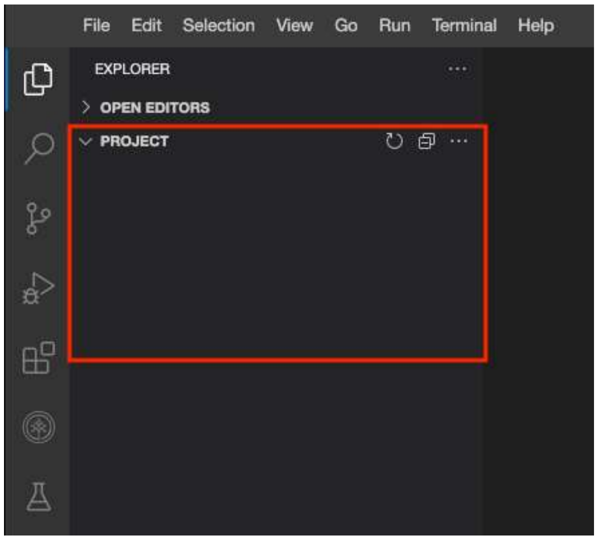
To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



If you have cloned (using `git clone` command) boilerplate/starting code, then it will look like below:

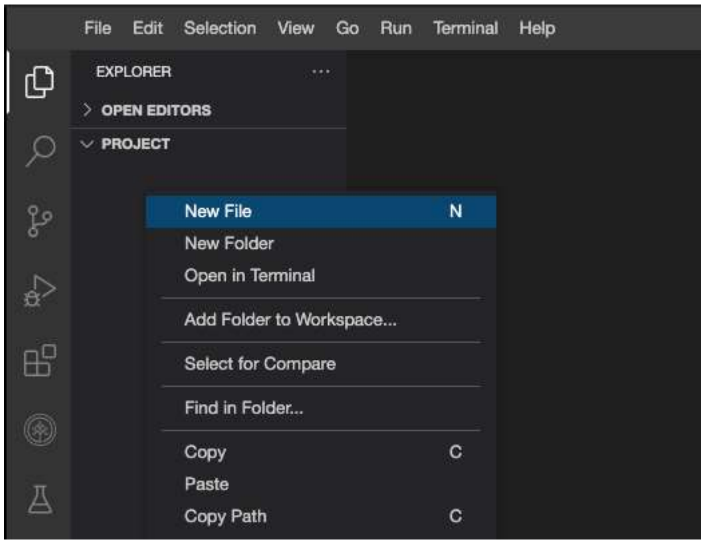


Otherwise a blank project looks like this:



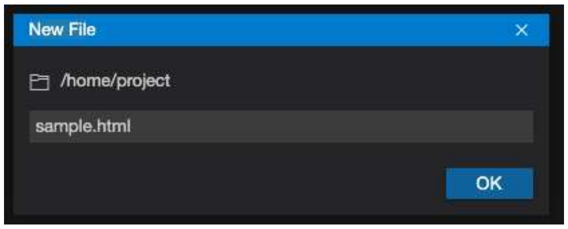
Create a new file

You can right-click and select the New File option to create a file in your project.

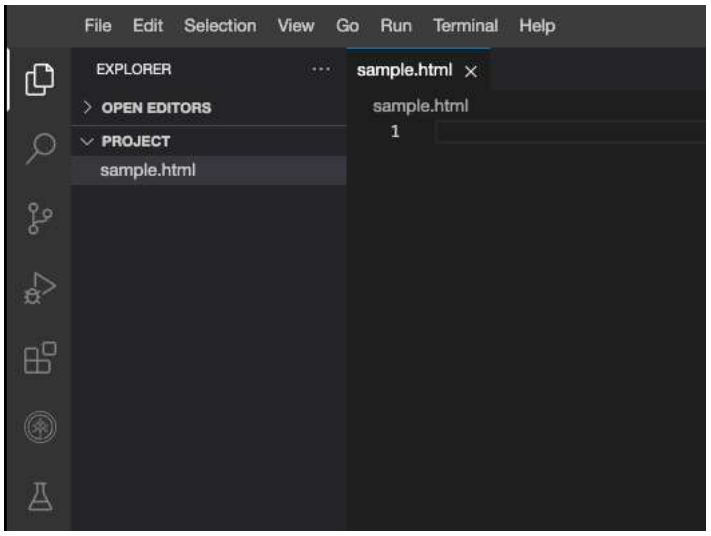


You can also choose `File -> New File` to do the same.

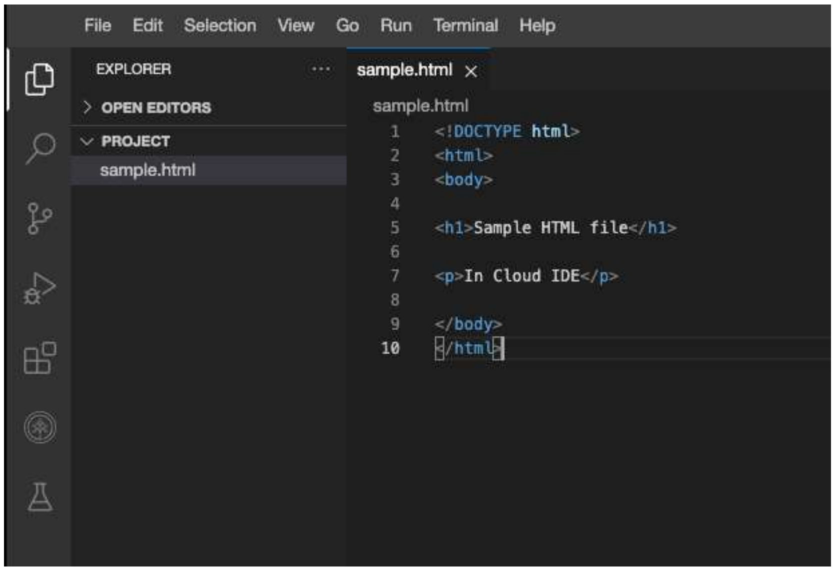
It will then prompt you to enter name of this new file. In the example below, we are creating `sample.html`.



Clicking on the file name `sample.html` in the directory structure will open the file on the right pane. You can create all different types of files; for example `FILE_NAME.js` for JavaScript file.



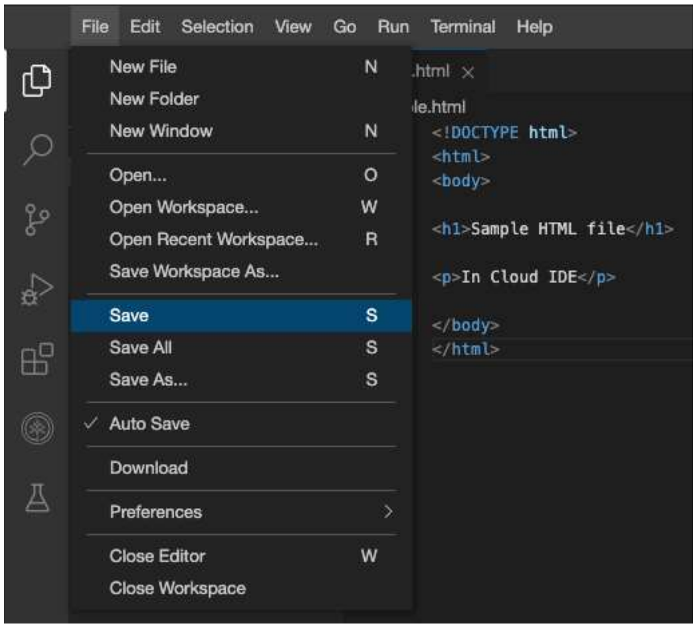
In the example, we just pasted some basic html code and then saved the file.



```
<!DOCTYPE html>
<html>
<body>

<h1>Sample HTML file</h1>

<p>In Cloud IDE</p>

</body>
</html>
```

And saving it by:

- Going in the menu.
- Press ⌘ `Cmd + S` on Mac or `CTRL + S` on Windows.
- Or it can Autosave it for you too.

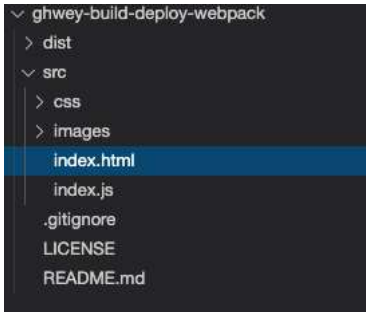


Project Structure

We have restructured the HTML project and keep `src` (for source) and `dist` (for output) directories.

Like before, `src` has `images` and `css` directories for our assets.

`dist` directory is empty at the moment.



```
ghwey-build-deploy-webpack
  dist
  src
    css
    images
    index.html
    index.js
  .gitignore
  LICENSE
  README.md
```

Exercise 1 : Add dependencies

We start by installing the required packages that will help us implement Webpack. Run the following command in the Terminal.

```
npm install webpack webpack-cli html-webpack-plugin --save-dev
```

Copied! Executed!

Where:

- `webpack` is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or asset.
- `webpack-cli` provides a flexible set of commands for developers to increase speed when setting up a custom Webpack project.
- `html-webpack-plugin` simplifies the creation of HTML files to serve the Webpack bundles.

You will also notice two new files created in your directory structure `package.json` and `package-lock.json`.

- `package.json` holds important information about the project. It contains both human-readable metadata about the project, such as the project name and description, and the functional metadata, such as the package version number and a list of dependencies required by the application.

- `package-lock.json` is automatically generated for any operations where npm modifies either the `node_modules` tree or `package.json`. It describes the exact tree that was generated, allowing subsequent installs to generate identical trees regardless of intermediate dependency updates.

Exercise 2 : Run Webpack for the first time

Let's run the following command, which will take our script at `src/index.js` as the entry point and will generate `dist/main.js` as the output. The `npx` command, which ships with Node 8.2/npm 5.2.0 or higher, runs the Webpack binary (./node_modules/.bin/webpack) of the Webpack package we installed in the beginning.

```
npx webpack
```

Copied! Executed!

Let's host the output from Webpack using python server. First, you need to go to the `dist` directory (in the Terminal).

```
cd dist
python3 -m http.server
```

Copied! Executed!

But where is our output HTML?

Let's add the required configuration for Webpack!

Exercise 3 : Define Webpack configuration

At the root directory of our project, we need to create `webpack.config.js` with the following configuration:

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

module.exports = {
  plugins: [
    new HtmlWebpackPlugin({
      hash: true,
      title: "Fresh Fruit Marketplace",
      metaDesc:
        "Find the freshest and highest quality fruits from local and global sources at our fruit marketplace. Shop for a variety of seasonal and exot
      header: "Fruit Marketplace",
      template: "./src/index.html",
      filename: "index.html",
      inject: "body",
    }),
  ],
  mode: "development",
  output: {
    clean: true,
  },
  module: {
    rules: [
      {
        test: /\.(jpg)$/i,
        type: "asset/resource",
        generator: {
          outputPath: "images/",
          publicPath: "images/",
          filename: "[name][ext]",
        },
      },
      {
        test: /\.(css)$/i,
        type: "asset/resource",
        generator: {
          outputPath: "css/",
          publicPath: "css/",
          filename: "[name][ext]",
        },
      },
    ],
  },
};
```

Copied!

Plugins

Plugins are the backbone of Webpack. Webpack itself is built on the same plugin system that you use in your Webpack configuration!

HTML-Webpack-plugin

Name	Type	Default	Description
`hash`	`{Boolean}`	`false`	If `true`, it adds a unique `webpack` compilation hash to all included scripts and CSS files. This is useful for cache busting.
`title`	`{String}`	`Webpack App`	Used for the generated HTML document.
`metaDesc`	`{String}`	‘’	Enables adding meta description into HTML with a customized configuration.
`header`	`{String}`	‘’	Enables adding page description into HTML with a customized configuration.
`template`	`{String}`	‘’	`webpack` relative or absolute path to the template. If it exists, it will use `src/index.ejs` by default. For more information, please see the docs.
`filename`	`{String|Function}`	`'index.html'`	The file where the HTML will be written. `index.html` is the default name. You can also specify a subdirectory here (eg: `assets/admin.html`). The `[name]` placeholder will be replaced with the entry name. It can also be a function e.g. `(entryName) => entryName + '.html'`.
`inject`	`{Boolean|String}`	`true`	`true || 'head' || 'body' || false` Injects all assets into the given `template` or `templateContent`. When passing `'body'` all Javascript resources will be placed at the bottom of the body element. `'head'` will place the scripts in the head element. Passing `true` will add it to the head/body depending on the `scriptLoading` option. Passing `false` will disable automatic injections. See the inject:false example for more details.

Mode

Using the mode configuration option instructs Webpack to use its built-in optimizations appropriately.

Defaults to `production`, possible values are: `none`, `development`, and `production`.

Please see the docs for more details.

Output

The top-level `output` key contains a set of options that tell Webpack how and where to output bundles, assets, and anything else bundled or loaded with Webpack.

output.clean

If set to `true`, it will clean the output directory before emit.

Please see the docs for more details.

Module

These options determine how the different types of modules within a project will be treated.

module.rules

An array of Rules which are matched to requests when modules are created. These rules can modify how the module is created. They can apply loaders to the module, or modify the parser.

Name	Type	Description
`test`	`{String}`	Includes all modules that pass test assertion.
`type`	`{String}`	Sets the type for a matching module. This prevents defaultRules and their default importing behaviours from occurring. For example, if you want to load a `.json` file through a custom loader, you'd need to set the type to `javascript/auto` to bypass Webpack's built-in json importing.
`generator.outputPath`	`{String}`	Emits the asset in the specified folder relative to `output.path`.
`generator.publicPath`	`{String}`	Customizes publicPath for asset modules.
`generator.filename`	`{String}`	`asset/resource` modules emit [hash][ext][query] filenames into output directory by default. This parameter overrides those values.

Exercise 4 : Inject values in index.html using Webpack

We will now need to import our assets (CSS and images) differently, and also make the required changes to inject other text values.

1. CSS import:

```
<link rel="stylesheet" href=<%=require('./css/main.css')%> type="text/css" media="all" />
```

Copied!

2. Title:

```
<title><%= htmlWebpackPlugin.options.title %></title>
```

Copied!

3. Meta description:

```
<meta name="description" content="<%= htmlWebpackPlugin.options.metaDesc %>" />
```

Copied!

4. Fruit Marketplace header:

```
<h1 class="text-center"><%= htmlWebpackPlugin.options.header %></h1>
```

Copied!

5. Fruit images:

```
<img src=<%=require('./images/banana.jpg')%> alt="...." />
```

Copied!

In `index.js` (`src` directory), add the following code:

```
console.log('Hello world!');
```

Copied!

Let's repackage it and host it. It is more convenient to open two terminals, one pointing to the project root directory and the other to the `dist` directory.

On the Terminal, running on root directory, run:

```
1. npx webpack
```

Copied! Executed!

And on the other one, run:

```
1. 1
```

```
1. python3 -m http.server
```

Copied! Executed!

Exercise 5 : Review output in dist directory

1. `main.js`: Webpack takes the source code from index.js file and bundles the final code here.
2. `index.html`: Similar to what we had before, but this time generated by Webpack. And we are able to inject values dynamically in our otherwise static HTML.
3. `images` (directory): Containing images that are used/referenced in our HTML page. You won't find `mangos.jpg` here.
4. `css` (directory): Similar to `images`, only containing those CSS files which are referenced.

Congratulations! You have completed the lab for Build and Deploy a Website Using Webpack.

Summary:

In this lab, you used Webpack to dynamically create a distributable from a static HTML website. This gives you an optimized package of your referenced assets as well as parameterized text.

Author(s)

Muhammad Yahya

Changelog

Date	Version	Changed by	Change Description
2023-03-23	0.2	Steve Hord	QA pass with edits
12-02-2023	0.1	Muhammad Yahya	Initial version created