

Task 4.5.-Compatibility with older browsers



Parcel

vs



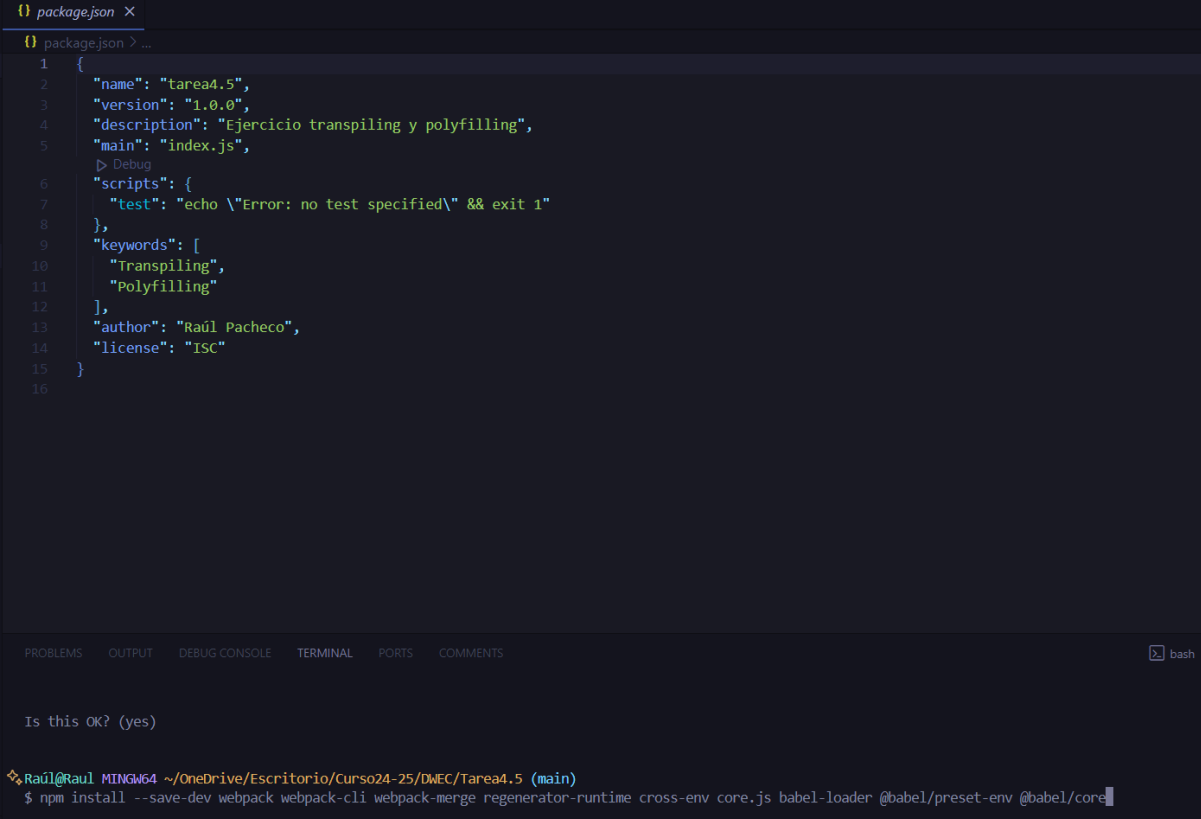
webpack

ÍNDICE

1. Webpack y Babel	3
2. Configuración para servir ficheros CSS	11
3. Parcel	15
4. Bibliografía	21
5. Repositorio GitHub	21

1. Webpack y Babel

Primero debemos instalar los siguientes paquetes utilizando node:

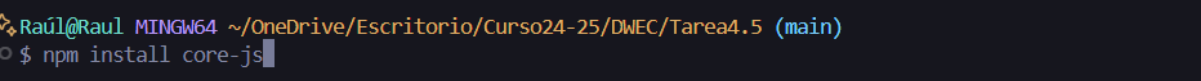


The screenshot shows a VS Code editor with a file named `package.json` open. The file contains the following JSON:

```
1 {
2   "name": "tarea4.5",
3   "version": "1.0.0",
4   "description": "Ejercicio transpiling y polyfilling",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [
10    "Transpiling",
11    "Polyfilling"
12  ],
13   "author": "Raúl Pacheco",
14   "license": "ISC"
15 }
```

Below the editor, the terminal shows the command `$ npm install --save-dev webpack webpack-cli webpack-merge regenerator-runtime cross-env core.js babel-loader @babel/preset-env @babel/core` being executed. The prompt "Is this OK? (yes)" is visible above the command.

También debemos instalar la siguiente dependencia pero sin usar `--save-dev` ya que se va usar en producción:



The screenshot shows a terminal window with the command `$ npm install core-js` being entered. The prompt is `Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5 (main)`.

Creamos los siguientes archivos: **babel.config.js**, **webpack.common.js**, **webpack.legacy.js** y **webpack.modern.js**.

✓ TAREA4.5

➤ node_modules

✓ src

➤ estilos

➤ html

➤ js

➤ media

🔗 index.html

📄 babel.config.js

📄 package-lock.json

📄 package.json

📄 webpack.common.js

📄 webpack.legacy.js

📄 webpack.modern.js

En el archivo **webpack.common.js** debemos incluir lo siguiente, donde **entry** es el archivo desde el cual Webpack comenzará a empaquetar tu código, **output** define cómo y dónde Webpack debe colocar los archivos generados después del proceso de bundling y mode le indica a Webpack si debe compilar en modo de desarrollo (development) o en modo de producción:

```
webpack.common.js X
webpack.common.js > ...
1  import path from 'path';
2
3
4  export default {
5    entry: './src/js/index.js',
6    output: {
7      path: path.resolve(process.cwd(), 'dist', process.env.mode),
8      filename: 'bundle.js',
9    },
10 };
11 |
```

El archivo **webpack.legacy.js** contendrá un conjunto de reglas para los archivos que Webpack debe procesar, **test** es un patrón que indica qué archivos deben ser procesados, en **exclude** se especifica que los archivos dentro de node_modules no deben ser procesados por esta regla y **use** define qué loader se debe usar para procesar los archivos. En este caso, se está utilizando babel-loader:

```
JS webpack.legacy.js U X
JS webpack.legacy.js > ...
2  import {merge} from 'webpack-merge';
3  import common from './webpack.common.js';
4
5
6  export default merge(common, {
7    output:{
8      filename: 'bundle.legacy.js',
9    },
10   module:{
11     rules:[
12       {
13         test: /\.js$/,
14         exclude: /node_modules/,
15         use: {
16           loader: 'babel-loader',
17         }
18       },
19     ],
20   },
21 }); |
```

En el archivo **webpack.modern.js** solo aparecerá **output** que es una propiedad que le indica a Webpack dónde y con qué nombre debe guardar el archivo final después de procesar todos los módulos.

```
JS webpack.modern.js U X
JS webpack.modern.js > ...
1  import path from 'path';
2  import {merge} from 'webpack-merge';
3  import common from './webpack.common.js';
4
5
6  export default merge(common, {
7    output:{
8      filename: 'bundle.modern.js',
9    },
10 });
11
```

Por último, el archivo **babel.config.js** que debe contener los **presets** que es una lista de conjuntos de reglas o configuraciones predefinidas que Babel usará para transpilar el código. **targets** define los navegadores o entornos que Babel debe soportar al transpilar el código. **corejs** especifica la versión de core-js que Babel debe usar para los polyfills. **useBuiltIns** optimiza la inclusión de polyfills (fragmentos de código que simulan funciones modernas en navegadores antiguos).

```
B babel.config.js U X
B babel.config.js > default
1  export default{
2    presets: [
3      [
4        '@babel/preset-env',{
5          targets: '> 0.25%, firefox>10, not dead', // Navegadores que van a ser compatibles con tu programa
6          useBuiltIns: 'usage',
7          corejs: 3
8        }
9      ]
10 ]
11 };
```

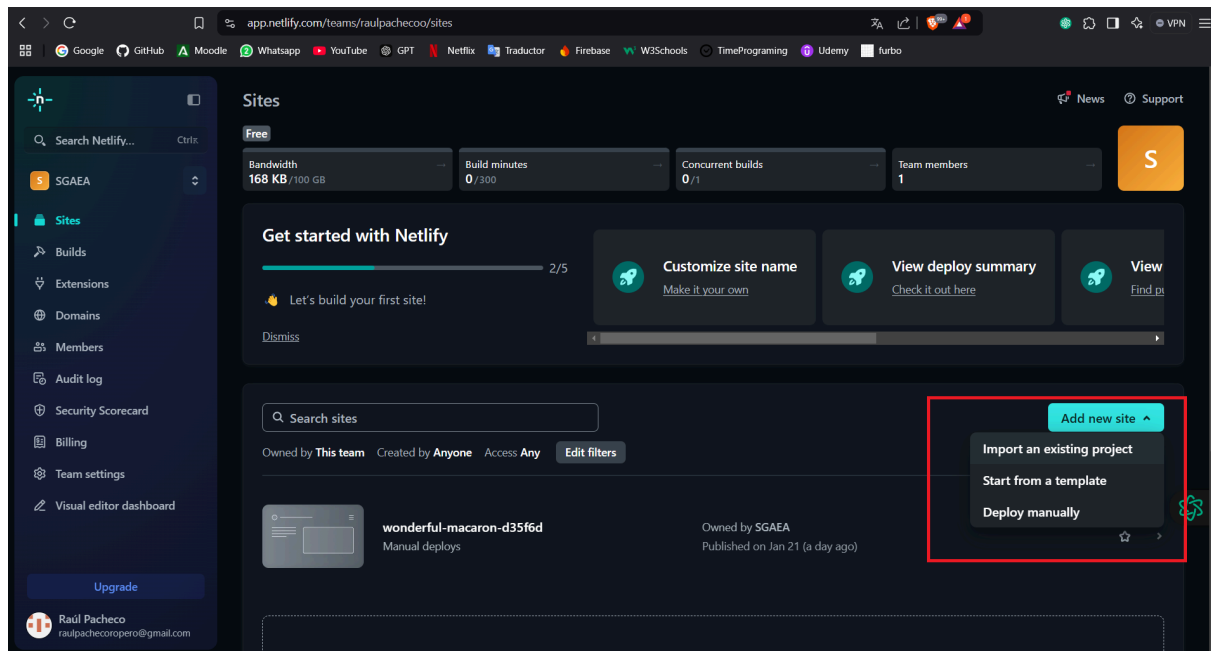
En el archivo **package.json** debemos definir los scripts que vamos a ejecutar:

```

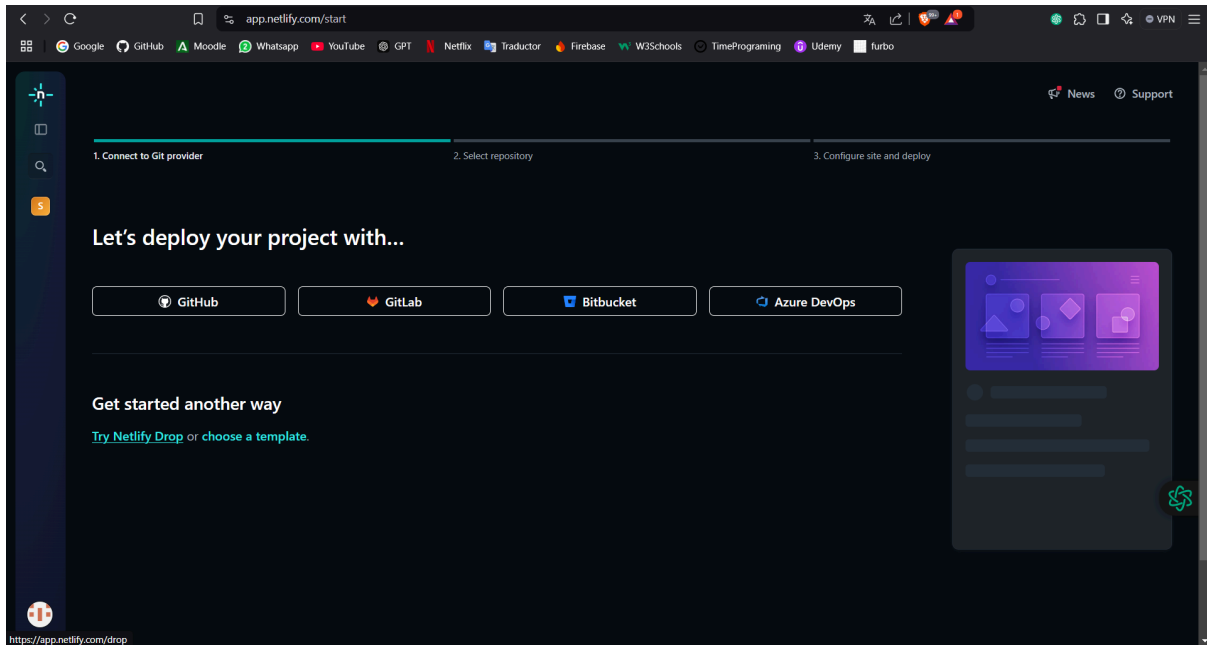
"scripts": {
  "build:legacy": "webpack --config webpack.legacy.js --mode %modo%",
  "build:modern": "webpack --config webpack.modern.js --mode %modo%",
  "des": "cross-env modo=development run-s build:legacy build:modern",
  "prod": "cross-env modo=production run-s build:legacy build:modern",
  "documenta": "npx jsdoc -c jsdoc.json",
  "limpia": "rimraf documentacion dist",
  "todo": "npm-run-all limpia documenta des"
},

```

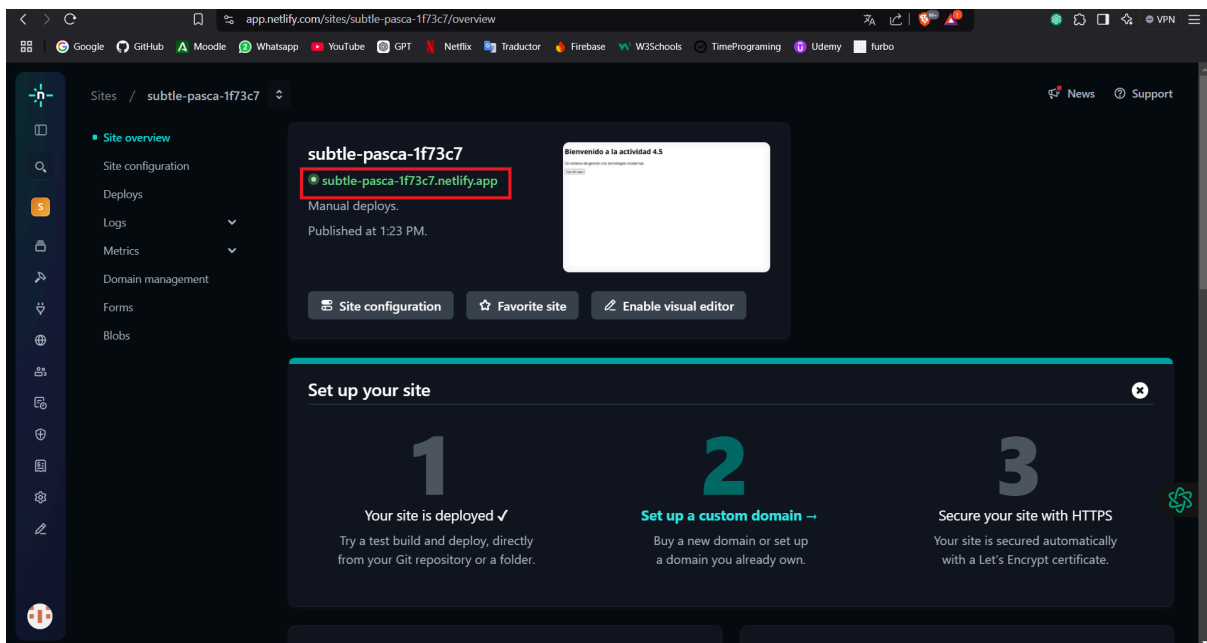
Para poder probar nuestro código, vamos a subir nuestro código a un proveedor de hosting como puede ser <https://www.netlify.com/>. Debemos acceder a esta página y crearnos una cuenta. Una vez que iniciamos sesión y estamos en la página de inicio, pulsamos en **Add new site y Import an existing project**:



Ahora seleccionamos **Try Netlify Drop** y subimos nuestro proyecto desde local y ya lo tenemos subido:



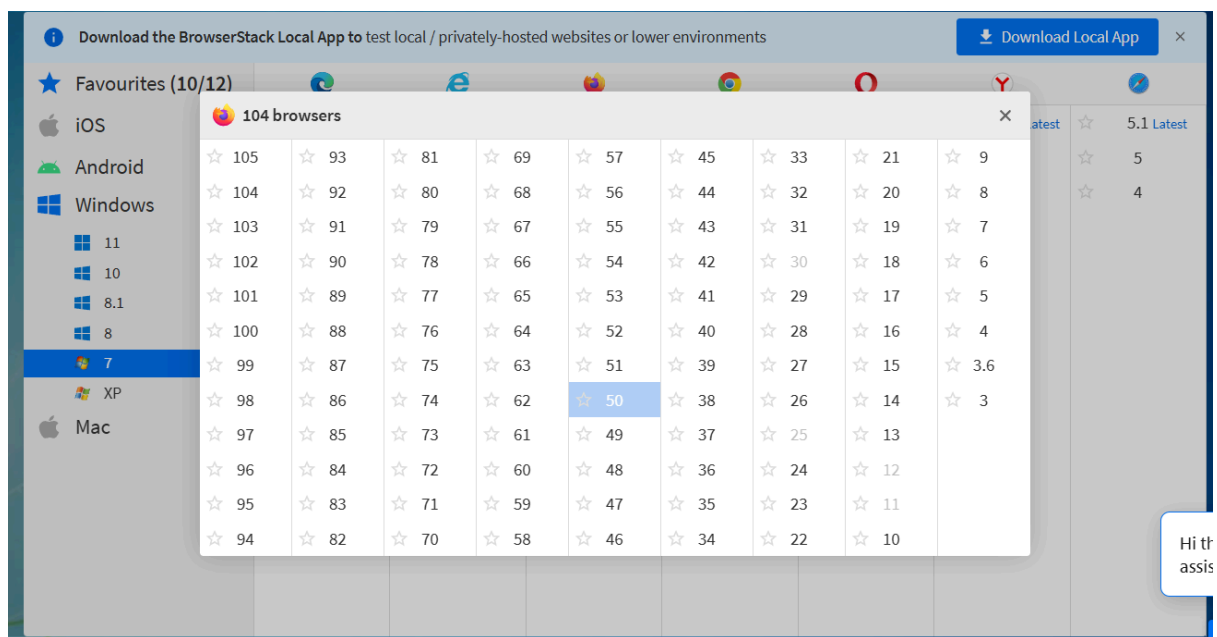
Para probarlo copiamos el enlace que nos proporciona:



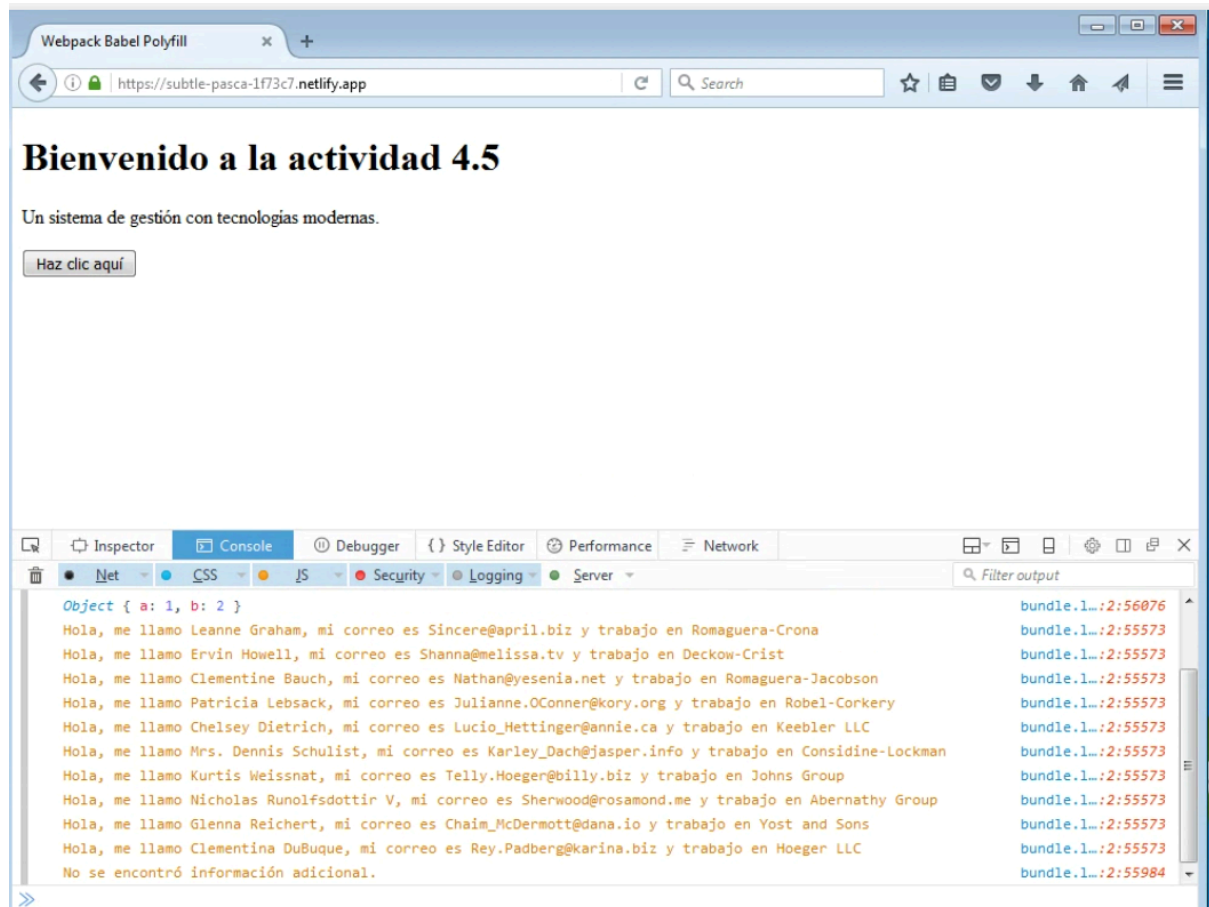
Debemos añadir lo siguiente a nuestro html para que el navegador elija el archivo js a ejecutar según la versión:

```
<script defer type="module" src="dist/production/bundle.modern.js"></script>
<script defer src="dist/production/bundle.legacy.js"></script>
```

Nos dirigimos a <https://www.browserstack.com/> donde nos registramos y vamos a elegir un navegador para probar nuestro código. Yo por ejemplo voy a probar **Firefox(versión 50)** ya que tiene una cuota de mercado mayor al 0,25% y una versión mayor a la 10.



Podemos comprobar que funciona correctamente nuestro programa.



2. Configuración para servir ficheros CSS

Primero debemos instalar las siguientes dependencias:

```
Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5 (main)
$ npm install --save-dev style-loader css-loader mini-css-extract-plugin

added 15 packages, and audited 434 packages in 2s

116 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ahora vamos a modificar los diferentes archivos que ya tenemos. Vamos a empezar por el **webpack.common.js**: En este archivo hay que añadir unas reglas para extraer el css de producción e inyectarlo en desarrollo, además también hay que añadir plugins poder realizar las tareas anteriormente comentadas.

```
webpack.common.js X
webpack.common.js > ...
1  import path from 'path';
2  import MiniCssExtractPlugin from 'mini-css-extract-plugin';
3
4  export default {
5    entry: './src/js/index.js',
6    output: {
7      path: path.resolve(process.cwd(), 'dist', process.env.mode),
8      filename: 'bundle.js', // Asegúrate de que la salida de JS también tenga un nombre
9    },
10   module: {
11     rules: [
12       // Regla para procesar archivos CSS
13       {
14         test: /\.css$/i, // Aplica esta regla a los archivos .css
15         use: [
16           process.env.NODE_ENV === 'production'
17             ? MiniCssExtractPlugin.loader // Extrae CSS en producción
18             : 'style-loader', // Inyecta CSS en el DOM en desarrollo
19           'css-loader', // Resuelve @import y url() dentro de CSS
20         ],
21       },
22     ],
23   },
24   plugins: [
25     // Plugin para extraer el CSS en un archivo separado en producción
26     process.env.NODE_ENV === 'production' &&
27     new MiniCssExtractPlugin({
28       filename: '[name].[contenthash].css', // Nombre del archivo CSS generado
29     }),
30   ].filter(Boolean), // Filtra el plugin si no estamos en producción
31   mode: process.env.mode || 'development', // Usar 'modo' del entorno, por defecto es 'development'
32 };
33 |
```

En los archivos **webpack.modern.js** y **webpack.legacy.js** tenemos que añadir el mismo código que hemos añadido a **webpack.common.js**.

```
JS webpack.modern.js U X
JS webpack.modern.js > ...
1 import path from 'path';
2 import { merge } from 'webpack-merge';
3 import common from './webpack.common.js';
4 import MiniCssExtractPlugin from 'mini-css-extract-plugin';
5
6 export default merge(common, {
7   output: {
8     filename: 'bundle.modern.js',
9   },
10  module: {
11    rules: [
12      {
13        test: /\.js$/,
14        exclude: /node_modules/,
15        use: {
16          loader: 'babel-loader',
17        }
18      },
19      // Añadimos la regla para procesar archivos CSS
20      {
21        test: /\.css$/i,
22        use: [
23          process.env.NODE_ENV === 'production'
24            ? MiniCssExtractPlugin.loader // Extrae el CSS en producción
25            : 'style-loader', // Inyecta el CSS en desarrollo
26          'css-loader', // Resuelve los imports y URLs en CSS
27        ],
28      },
29    ],
30  },
31  plugins: [
32    // Plugin para extraer el CSS en un archivo separado en producción
33    process.env.NODE_ENV === 'production' &&
34    new MiniCssExtractPlugin({
35      filename: '[name].[contenthash].css',
36    }),
37  ].filter(Boolean),
38 });
39
```

```

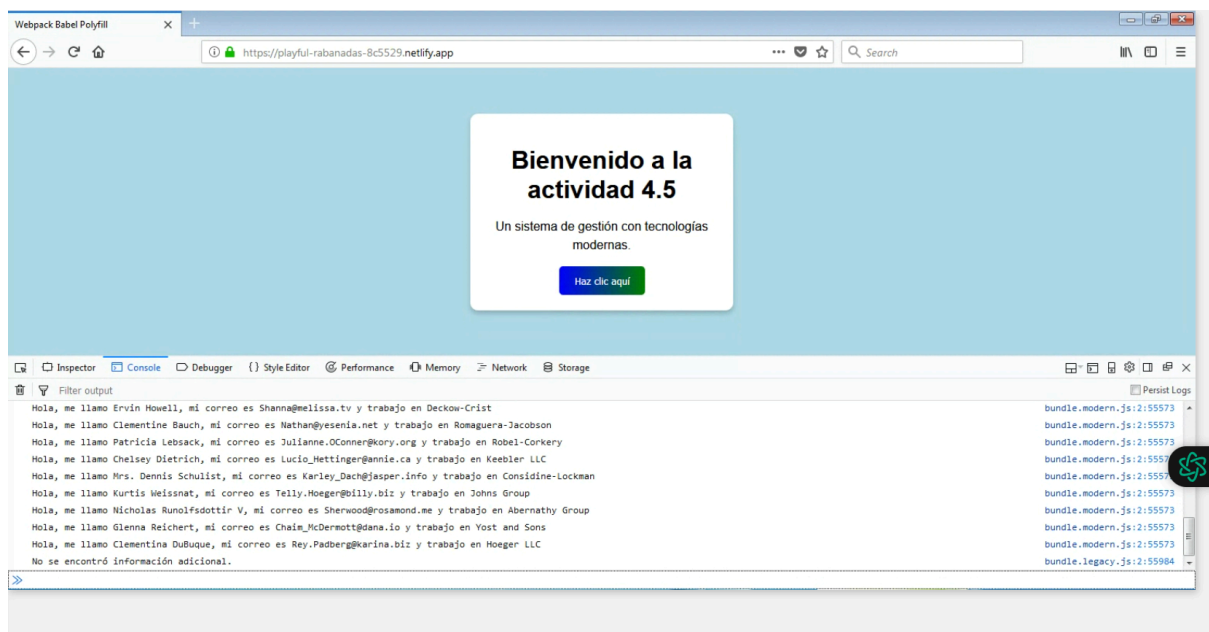
JS webpack.legacy.js U X
JS webpack.legacy.js > ...
1  import path from 'path';
2  import { merge } from 'webpack-merge';
3  import common from './webpack.common.js';
4  import MiniCssExtractPlugin from 'mini-css-extract-plugin';
5
6  export default merge(common, {
7    output: {
8      filename: 'bundle.legacy.js',
9    },
10   module: {
11     rules: [
12       {
13         test: /\.js$/,
14         exclude: /node_modules/,
15         use: {
16           loader: 'babel-loader',
17         }
18       },
19       // Añadimos la regla para procesar archivos CSS
20       {
21         test: /\.css$/i,
22         use: [
23           process.env.NODE_ENV === 'production'
24             ? MiniCssExtractPlugin.loader // Extrae el CSS en producción
25             : 'style-loader', // Inyecta el CSS en desarrollo
26           'css-loader', // Resuelve los imports y URLs en CSS
27         ],
28       },
29     ],
30   },
31   plugins: [
32     // Plugin para extraer el CSS en un archivo separado en producción
33     process.env.NODE_ENV === 'production' &&
34     new MiniCssExtractPlugin({
35       filename: '[name].[contenthash].css',
36     }),
37   ].filter(Boolean),
38 });
39

```

Vamos a añadir código css que no soportan los navegadores antiguos.

```
# estilos.css
src > estilos > # estilos.css > ...
1 body {
2   background-color: lightblue; /* Funciona en casi todos los navegadores */
3   display: grid; /* No funciona en IE 10 y anteriores */
4   place-items: center; /* No funciona en IE 10 y anteriores */
5   height: 100vh;
6   margin: 0;
7   font-family: Arial, sans-serif;
8 }
9 .container {
10  background-color: white;
11  padding: 20px;
12  border-radius: 10px; /* No funciona en IE 8 y anteriores */
13  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2); /* No funciona en IE 8 y anteriores */
14  text-align: center;
15  width: 300px;
16 }
17 h1 {
18   font-size: 2rem; /* Las unidades rem no funcionan en IE 8 y anteriores */
19   margin-bottom: 10px;
20 }
21 p {
22   font-size: 1rem;
23   line-height: 1.5; /* No funciona en IE 8 y anteriores */
24 }
25 button {
26   background: linear-gradient(to right, blue, green); /* No funciona en IE 9 y anteriores */
27   color: white;
28   border: none;
29   padding: 10px 20px;
30   border-radius: 5px;
31   cursor: pointer;
32   transition: transform 0.3s ease; /* No funciona en IE 9 y anteriores */
33 }
34
35 button:hover {
36   transform: scale(1.1); /* No funciona en IE 9 y anteriores */
37 }
38
```

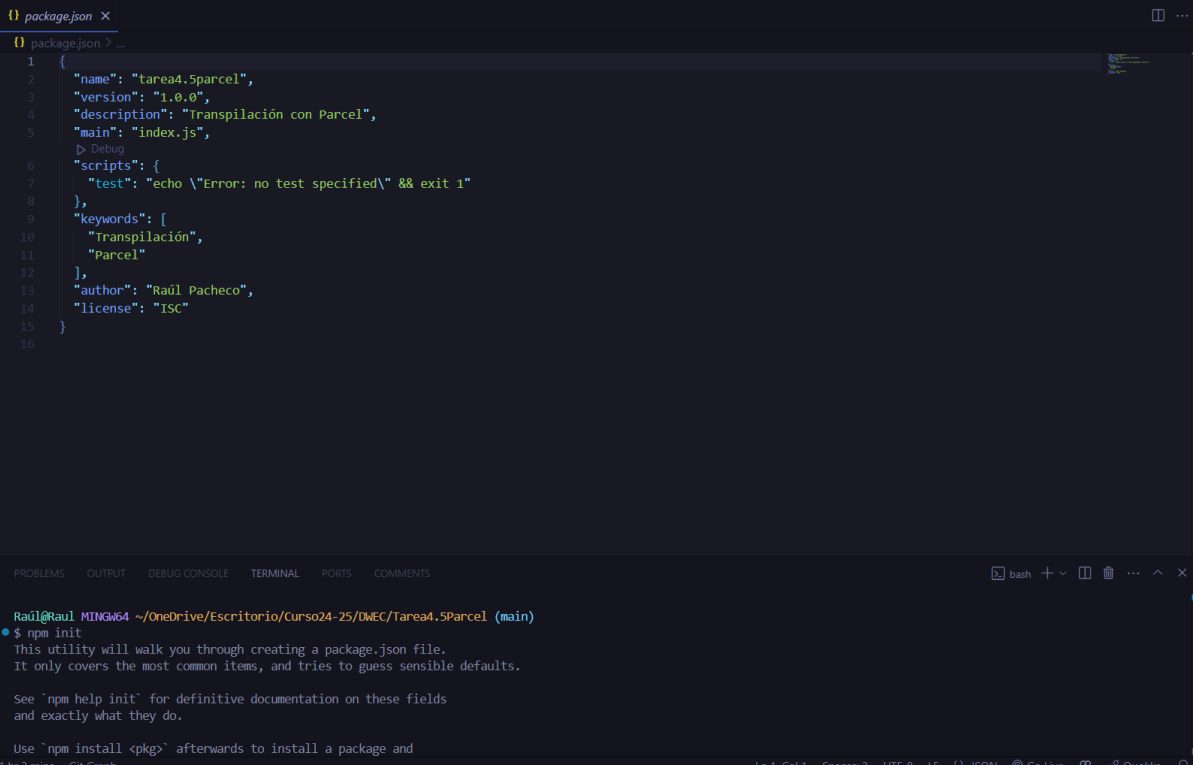
Los demás archivos no cambian. Ahora vamos a proceder a probar esto en Firefox(versión 60).



3.Parcel

Para realizar la transpiración con Parcel primero vamos a crearnos un nuevo proyecto con la siguiente estructura y a iniciar node para que se cree el archivo package.json:

```
project/
├── src/
│   ├── index.html
│   └── estilos
│       └── estilos.css
├── js
│   └── index.js
└── package.json
```



The screenshot shows a Visual Studio Code editor with a file named `package.json` open. The file contains the following JSON content:

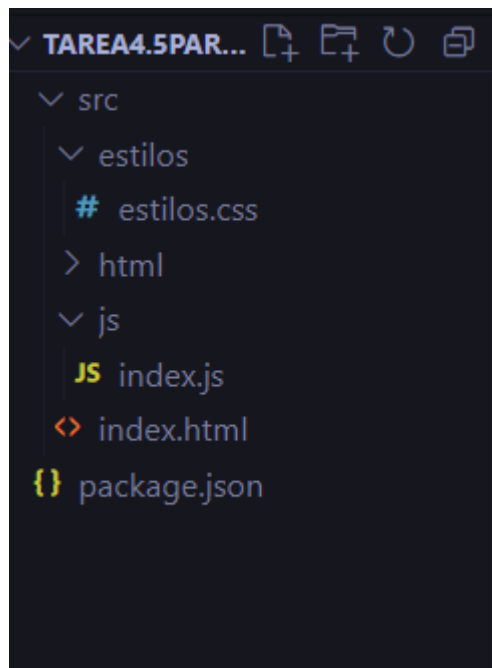
```
{
  "name": "tarea4.5parcel",
  "version": "1.0.0",
  "description": "Transpilación con Parcel",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [
    "Transpilación",
    "Parcel"
  ],
  "author": "Raúl Pacheco",
  "license": "ISC"
}
```

Below the editor, the terminal window shows the output of the `npm init` command:

```
Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5Parcel (main)
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
```



Ahora vamos a insertar un código de ejemplo en los archivos:

index.js



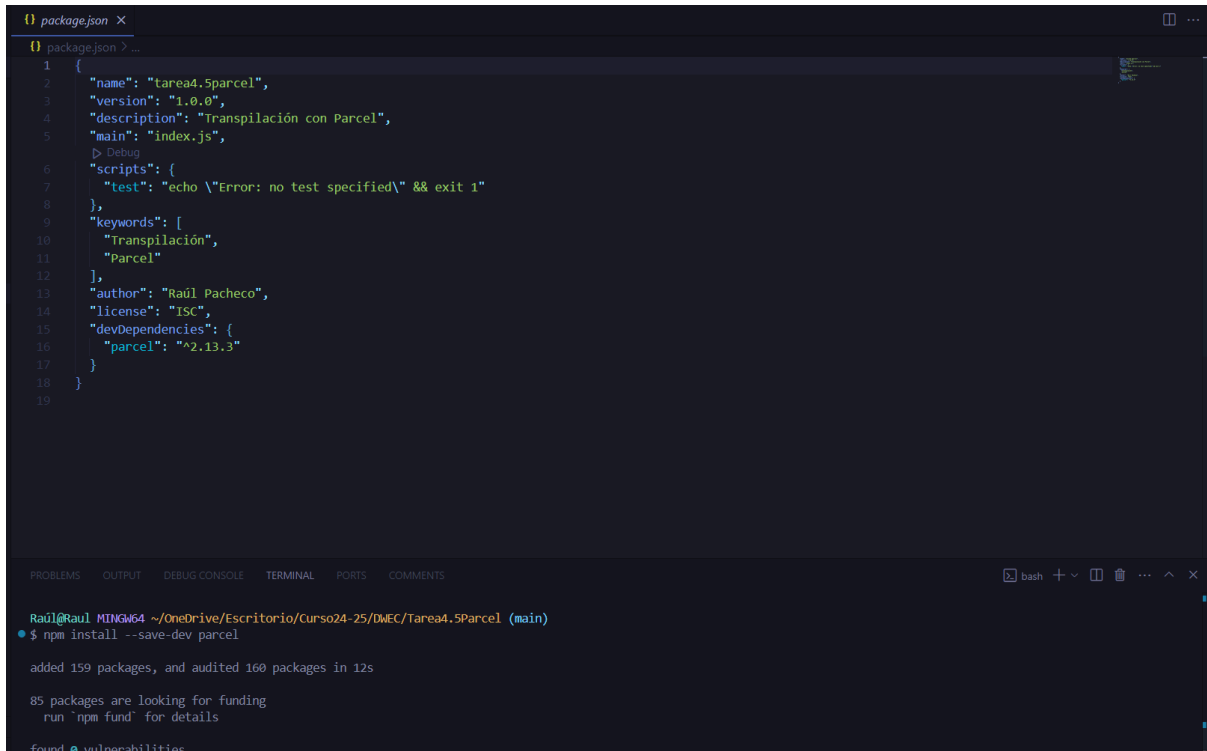
index.html

```
index.html M X
Tarea4.5Parcel > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Parcel</title>
7   <link rel="stylesheet" href="./src/estilos/estilos.css">
8 </head>
9 <body>
10   <div class="container">
11     <h1>Bienvenido a nuestro proyecto con Parcel</h1>
12     <button>Haz clic aquí</button>
13   </div>
14   <script src="./src/js/index.js"></script>
15 </body>
16 </html>
17
```

estilos.css

```
# estilos.css X
src > estilos > # estilos.css > ...
1 body {
2   background-color: red;
3   display: flex;
4   justify-content: center;
5   align-items: center;
6   height: 100vh;
7   margin: 0;
8   font-family: Arial, sans-serif;
9 }
10
11 .container {
12   text-align: center;
13   padding: 20px;
14   border-radius: 8px;
15   background-color: white;
16   box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);
17 }
18
19 button {
20   padding: 10px 20px;
21   background-color: blue;
22   color: white;
23   border: none;
24   border-radius: 5px;
25   cursor: pointer;
26 }
27
28 button:hover {
29   background-color: darkblue;
30 }
31
```

Debemos instalar la dependencia de Parcel:



```
1 {
2   "name": "tarea4.5parcel",
3   "version": "1.0.0",
4   "description": "Transpilación con Parcel",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\Error: no test specified\\ && exit 1"
8   },
9   "keywords": [
10    "Transpilación",
11    "Parcel"
12  ],
13  "author": "Raúl Pacheco",
14  "license": "ISC",
15  "devDependencies": {
16    "parcel": "^2.13.3"
17  }
18 }
19
```

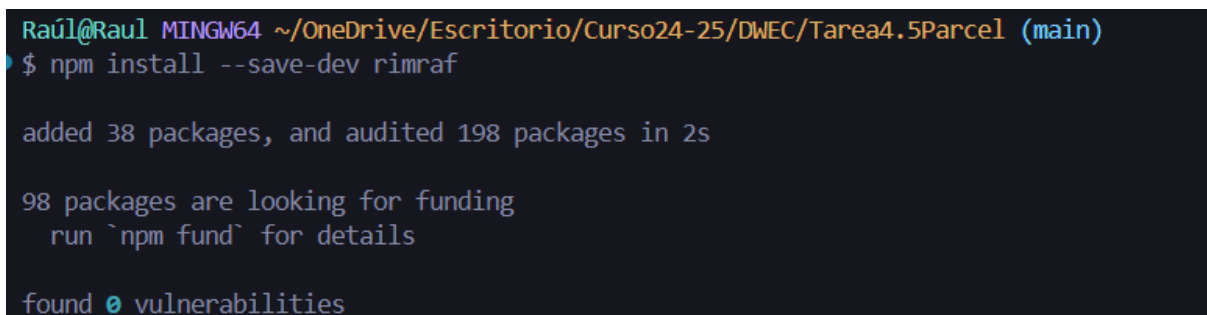
```
Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5Parcel (main)
$ npm install --save-dev parcel

added 159 packages, and audited 160 packages in 12s

85 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Vamos a instalar **rimraf** también para que nos permita eliminar los archivos transpilados.



```
Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5Parcel (main)
$ npm install --save-dev rimraf

added 38 packages, and audited 198 packages in 2s

98 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Una vez instalada esta dependencia, procedemos a configurar los scripts en el archivo **package.json**



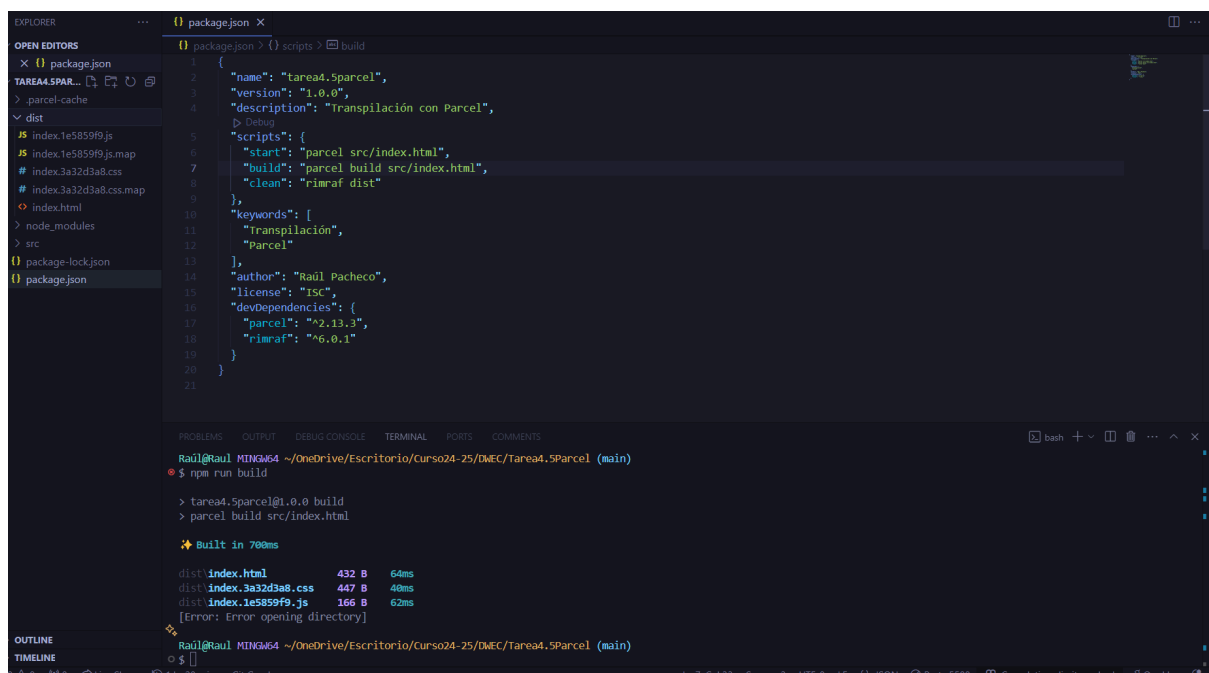
```
"scripts": {
  "start": "parcel src/index.html",
  "build": "parcel build src/index.html",
  "clean": "rimraf dist"
},
```

Vamos a añadir también los navegadores objetivo en nuestro archivo **package.json**, aunque por defecto Parcel soportará los siguientes navegadores:

- Los últimos 2 navegadores principales.
- Navegadores con más del 0.5% de cuota de mercado.
- Navegadores que no estén marcados como "dead".

```
"browserslist": [  
  "> 0.25%",  
  "not dead",  
  "firefox >= 20"  
],  
  "Debug"
```

Ahora vamos a proceder a ejecutarlos. Debemos ejecutar el comando **npm run build**.



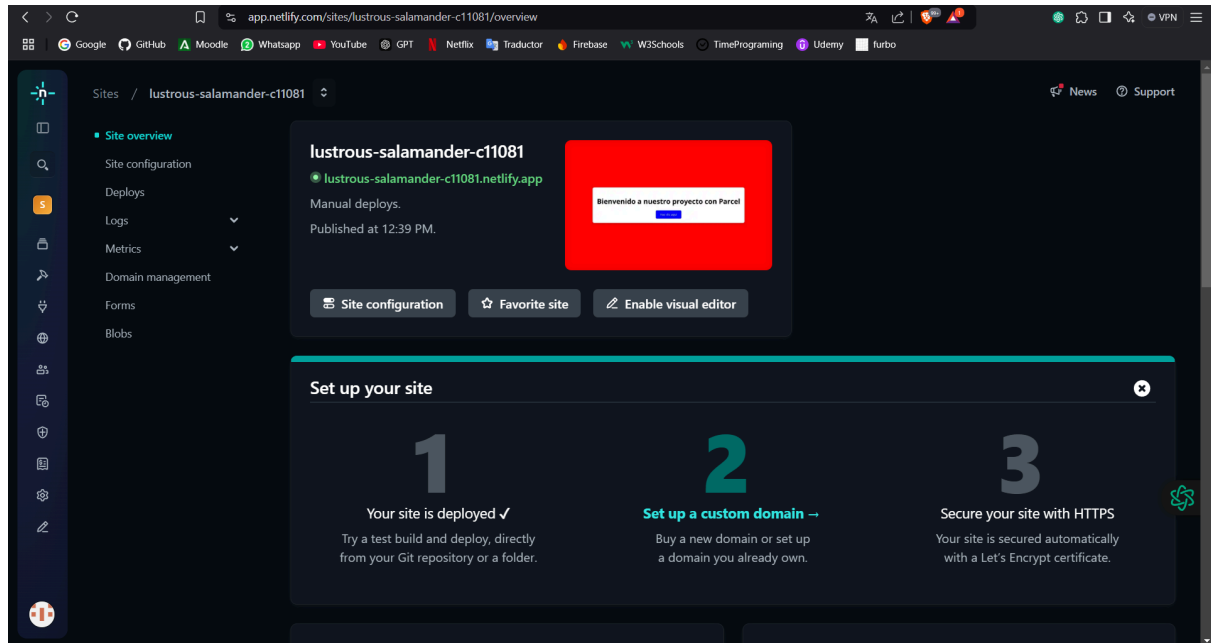
The screenshot shows the VS Code interface with the `package.json` file open in the editor. The file contains the following content:

```
{  
  "name": "tarea4.5parcel",  
  "version": "1.0.0",  
  "description": "Transpilación con Parcel",  
  "scripts": {  
    "start": "parcel src/index.html",  
    "build": "parcel build src/index.html",  
    "clean": "rimraf dist"  
  },  
  "keywords": [  
    "transpilación",  
    "Parcel"  
  ],  
  "author": "Raúl Pacheco",  
  "license": "ISC",  
  "devDependencies": {  
    "parcel": "^2.13.3",  
    "rimraf": "^6.0.1"  
  }  
}
```

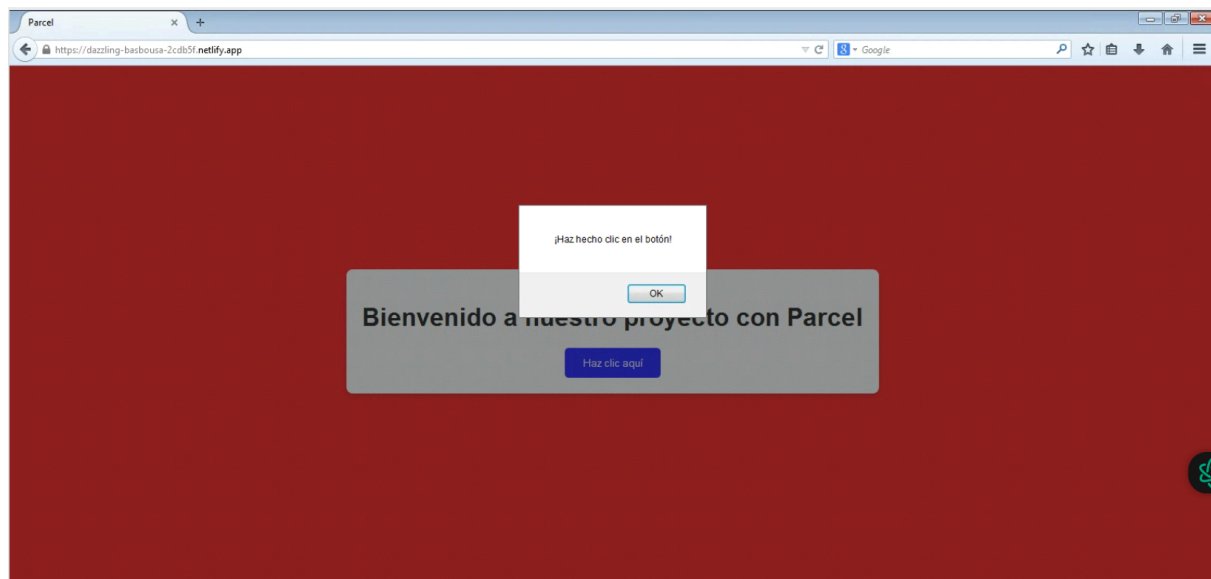
The terminal window at the bottom shows the execution of the `npm run build` command. The output is as follows:

```
Raúl@Raúl MINGW64 ~/OneDrive/Escritorio/Curso24-25/DWEC/Tarea4.5Parcel (main)  
$ npm run build  
  
> tarea4.5parcel@1.0.0 build  
> parcel build src/index.html  
  
✨ Built in 700ms  
  
dist\index.html      432 B   64ms  
dist\index.3a32d3a8.css 447 B   40ms  
dist\index.1e5859f9.js  166 B   62ms  
[Error: Error opening directory]
```

Vamos a subir nuestro proyecto a **netlify**, siguiendo el proceso anteriormente comentado para poder probarlo.



Y vamos a probarlo en **browserstack** usando por ejemplo **Firefox(versión 30)**



4. Bibliografía

<https://pablomagaz.com/blog/empaquetando-aplicaciones-con-parcel/>

<https://www.youtube.com/watch?v=YnZ883a7XP0>

5. Repositorio GitHub

En el siguiente enlace se encuentra el código de las dos actividades:

- **Actividad4.5**
- **Actividad4.5Parcel**

La documentación de la actividad también está en el Readme.md del repositorio.

https://github.com/RaulPachecoo/DWEC_Raul_Pacheco_Ropero