A. INTRODUCCION A NODE JS

1 NODEJS

Node.js es un entorno en tiempo de ejecución multiplataforma para la capa del servidor (en el lado del servidor) basado en JavaScript.

Node.js es un entorno controlado por eventos diseñado para crear aplicaciones escalables, permitiéndote establecer y gestionar múltiples conexiones al mismo tiempo. Gracias a esta característica, no tienes que preocuparte con el bloqueo de procesos, pues no hay bloqueos.

Node.js se ha hecho popular en los últimos años gracias a las siguientes características:

- **Velocidad**. Node.js está construido sobre el motor de JavaScript V8 de Google Chrome, por eso su biblioteca es muy rápida en la ejecución de código.
- **Sin búfer**. Las aplicaciones de Node.js generan los datos en trozos (chunks), nunca los almacenan en búfer.
- **Asíncrono y controlado por eventos**. Como hemos dicho anteriormente, las APIs de la biblioteca de Node.js son asíncronas, sin bloqueo. Un servidor basado en Node.js no espera que una API devuelva datos. El servidor pasa a la siguiente API después de llamarla, y un mecanismo de notificación de eventos ayuda al servidor a obtener una respuesta de la llamada a la API anterior.
- **Un subproceso escalable**. Node.js utiliza un modelo de un solo subproceso con bucle de eventos. Gracias al mecanismo de eventos, el servidor responde sin bloqueos, como hemos dicho. Esto hace que el servidor sea altamente escalable comparando con los servidores tradicionales como el Servidor HTTP de Apache.

Puedes utilizar Node.js para diferentes tipos de aplicaciones. Los siguientes son algunos de los ejemplos:

- Aplicaciones de transmisión de datos (streaming)
- Aplicaciones intensivas de datos en tiempo real
- Aplicaciones vinculadas a E/S
- Aplicaciones basadas en JSON: API
- Aplicaciones de página única

Casi todas las marcas más importantes del mercado utilizan Node.js. A continuación, tienes algunos ejemplos:

- Microsoft
- eBay
- General Electric
- PayPal
- Uber
- NASA
- Netflix
- LinkedIn

2 TOOLING

Tooling es el término anglosajón usado para describir un conjunto de herramientas de forma genérica.

Cuando hablamos del tooling dentro del ecosistema de JavaScript nos referimos a todas aquellas utilidades auxiliares que nos servirán de apoyo y ayuda durante el desarrollo de nuestros proyectos.

Gran parte de las acciones que llevamos a cabo durante la fase de desarrollo son mecánicas y repetitivas. Tener un entorno o tooling que automatice estos procesos nos permitirá minimizar el tiempo que les dedicamos y centrarnos de manera más eficiente y efectiva en el desarrollo de nuestras aplicaciones.

En la historia moderna de la automatización de los entornos de desarrollo front-end se han quemado fases con protagonistas muy concretos, cronológicamente:

2.1 GRUNT

En términos generales, podemos decir que Grunt es un programa para correr tareas que han sido programadas en JavaScript. De forma más detallada, Grunt en JavaScript es una herramienta que te permitirá simplificar el proceso de construcción (build) de tus proyectos. Del mismo modo en que Maven se usa para automatizar una serie de pasos o procesos a la hora de compilar un código Java, Grunt permite lo mismo, pero en JavaScript. https://gruntjs.com/

2.2 GULP

Gulp es una herramienta, en forma de script en NodeJS, que te ayuda a automatizar tareas comunes en el desarrollo de una aplicación, como pueden ser: mover archivos de una carpeta a otra, eliminarlos, minificar código, sincronizar el navegador cuando modificas tu código, validar sintáxis y un largo etcétera. https://gulpjs.com/

2.3 WEBPACK

Nació en 2012 y en la actualidad es utilizado por miles de proyectos de desarrollo web Front-End: desde frameworks como React o Angular hasta en el desarrollo de aplicaciones tan conocidas como Twitter, Instagram, PayPal o la versión web de WhatsApp. Se define como un empaquetador de módulos (bundler) pero que hace muchísimas cosas más. Actualmente todo nuevo proyecto debe considerar **Webpack** como la opción por defecto. https://webpack.js.org/

2.4 PARCEL

En nuestro caso vamos a utilizar **Parcel**, una herramienta muy similar a Webpack que proporciona una serie de herramientas que requieren mucha menos configuración que este pero que ofrecen la misma flexibilidad y potencia.

Parcel es una herramienta Javascript que procesa todo el código de nuestro sitio o aplicación web, recorriendo todos sus archivos enlazados, y generando una nueva colección de ficheros, más apropiados para el navegador.

De esta forma, Parcel permite partir de un fichero inicial (HTML o Javascript, por ejemplo) e ir revisando todos los archivos que se utilizan y generando en una nueva carpeta (normalmente dist) todos los archivos referenciados, no teniendo en cuenta los que no se estén utilizando.

Además, Parcel también soporta escribir código en otros lenguajes o preprocesadores, convirtiendo al lenguaje necesario final, lo que lo hace su utilización mucho más sencilla que con otros sistemas.

Lo que nos ofrece Parcel (entre otras cosas) es:

- Live server para nuestras aplicaciones
- Hot reload de nuestros cambios
- Bundling rápido de JS
- Poder usar módulos de ES6 (import/export)

Todo esto puede no significar mucho ahora mismo, pero volveremos a estos puntos más adelante según empecemos a desarrollar y aclararemos el significado de cada uno de ellos

3 NODE.JS Y NPM

Históricamente, JavaScript había sido un lenguaje ejecutado en el navegador. Los navegadores son aplicaciones de escritorio que albergan en su interior un motor (engine) que compila y ejecuta JavaScript.

Sin embargo, Ryan Dahl cambió este paradigma el 8 de noviembre de 2009 con su trabajo y presentación de node.js en la JSConf.

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome, dicho de modo muy simple, una emulación de lo que sucede en el navegador sin el renderizado visual (no existe el objeto window).

Este hecho fue disruptivo, ya que permitió que JavaScript, uno de los lenguajes de programación más populares y con una curva de entrada más baja, pasara a ser, simultáneamente, un lenguaje de cliente y de servidor, lo que permitía que muchos ingenieros de front-end pudieran dar el salto a back-end sin tener que cambiar de lenguaje.

Node.js permitió el desarrollo de muchas aplicaciones no destinadas directamente a los navegadores y facilitó la aparición de un ecosistema de utilidades a las que se les llama paquetes (packages).

Al instalar node.js en nuestro ordenador se instala también un gestor de paquetes: el npm, node package manager (gestor de paquetes de node). Esta utilidad nos permite (entre otras cosas):

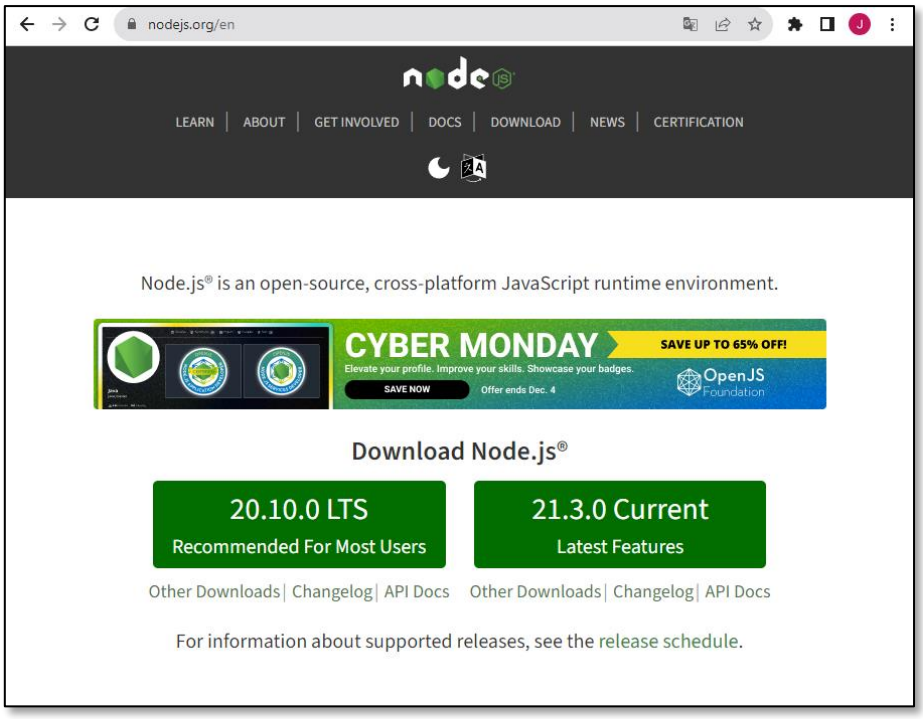
- Crear nuestros propios paquetes (nuestras aplicaciones).
- Instalar y usar paquetes desarrollados por terceros en nuestra aplicación.

Para poder tener una idea del tipo de aplicaciones a las que nos referimos podemos pasarnos por npmjs.com, el repositorio de paquetes oficial de node.js, una especie de supermercado de utilidades.

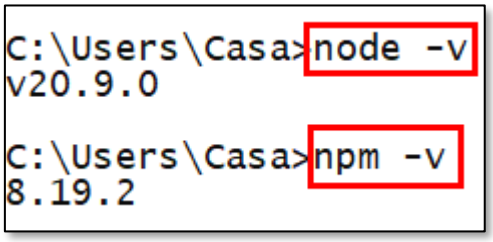
3.1 INSTALACIÓN DE NODE.JS Y NPM

Ahora que sabemos qué es y para qué sirven node.js y npm, podemos dirigirnos a su página oficial (https://nodejs.org/en) para descargarnos el binario más adecuado para nuestra plataforma de desarrollo.

Se nos presentan dos opciones: última versión o **LTS** (long term support); esta última es la opción más segura y aconsejable en nuestro caso, ya que no tenemos necesidad de hacer uso de las últimas novedades ni versiones experimentales con poco soporte.



Para comprobar que la instalación ha sido correcta, debemos abrir un terminal y ejecutar los siguientes comandos: ***node -v*** y ***npm -v***



La utilidad **npm** es lo que se conoce como una **interfaz de línea de comandos** (CLI) y es lo que nos permite importar código de terceros desde el registro de npm (https://www.npmjs.com/). Esto implica que el uso de la herramienta debe hacerse mediante la consola de nuestro sistema operativo.

4 CREANDO LA BASE DE LA APLICACIÓN

Nuestros ejercicios no serán necesariamente aplicaciones de escritorio. Sin embargo, el tooling alrededor de nuestra aplicación sí lo será.

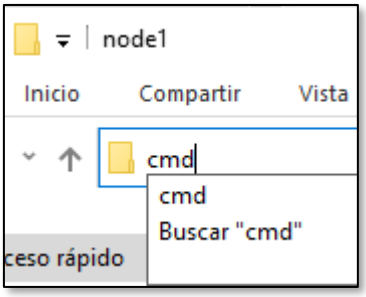
Queremos poder hacer uso de algunas de las funcionalidades que podemos encontrar en la librería de paquetes de npm para nuestro proyecto y, como dijimos, automatizar y optimizar nuestros procesos en fase de desarrollo.

Para arrancar la creación de nuestro proyecto, debemos buscar una carpeta en nuestro disco duro que nos parezca apropiado.

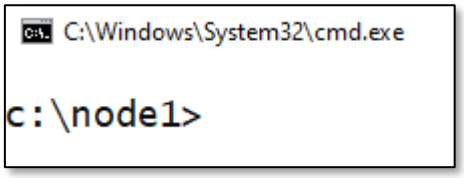
En mi caso crearé carpetas para cada uno de los ejercicios que desarrolle.

Una vez tengamos ubicada nuestra carpeta de proyectos, crearemos una nueva carpeta destinada a albergar nuestra aplicación y navegaremos hacia ella:

Dentro de la carpeta abriremos una ventana de cmd. La forma más rápida es escribir en la barra de direcciones cmd y pulsar intro.



Nos aparecerá la ventana del símbolo de sistema en esta carpeta.



Una vez en su interior, llamamos al inicializador de paquetes de node.js:

npm init

Podemos pasar por todas las preguntas que nos solicita el prompt con `enter` para aceptar los valores por defecto o directamente usar el comando con un parámetro adicional para obviarlas:

npm init -y

```
c:\node1>npm init -v
Wrote to D:\node1\package.json

{
  "name": "node1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
 exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}


c:\node1>
```

Esto que puede parecer tremendamente sofisticado únicamente genera un fichero llamado `package.json` en la raíz de nuestro proyecto con los campos y valores que hemos determinado.

Si echamos un vistazo a nuestro `**package.json**` veremos algo parecido a esto:

Esta es la firma de nuestro paquete. Todos los campos son autodescriptivos excepto dos, que pueden requerir algo más de explicación:

- `**main**`: es el punto de entrada de nuestra aplicación, normalmente `index.html` para una aplicación de navegador e `index.js` para una SPA y aplicaciones de escritorio o servidor.
- `**scripts**`: es un objeto en el que podemos definir todos los comandos asociados a nuestro proyecto y que podrán ser ejecutados desde la raíz usando el comando `**npm run <nombre_del_script>**`

Nota: Todo este proceso lanzado mediante `**npm init**` podría haberse hecho de forma manual creando el fichero `**package.json**` y definido en su interior el objeto con las propiedades de nuestro proyecto.

Aprovecharemos también para cambiar el punto de entrada de nuestra aplicación:

```
"main": "index.html",
```

4.1 INSTALANDO UNA DEPENDENCIA

Aunque hasta ahora solo tengamos una carpeta con un fichero de definición del paquete, lo que hemos habilitado, entre otras muchas cosas, es la posibilidad de instalar dependencias.

Una dependencia es un paquete externo alojado en un registro y desarrollado por terceros, de nuestro propio repositorio interno (como hacen muchas empresas) o bien local, de otro paquete/aplicación que hayamos desarrollado y almacenado en nuestro disco duro.

El primer paquete que vamos a instalar es Parcel.js usando el comando:

npm install --save-dev parcel-bundler

O su versión abreviada:

npm i -D parcel-bundler

```
c:\node1>npm i -D parcel-bundler
[                    ] / idealTree:@babel/core: ti
```

Esto dará paso a la instalación del paquete. Veremos una barra de proceso, en la que se descargará tanto `parcel` como sus propias dependencias, que quedarán instaladas en la carpeta `node_modules`. Aquí es donde residirán todas las dependencias de nuestro proyecto (y las subdependencias de nuestras dependencias).

```
en if nothing is polyfilled. Some versions have w
eb compatibility issues. Please, upgrade your dep
endencies to the actual version of core-js.

added 807 packages, and audited 808 packages in 2
m

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

56 vulnerabilities (44 moderate, 12 high)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

Todos los ficheros deben estar en local para poder hacer uso de estas utilidades de node. Sin embargo, si trabajamos de forma conjunta con otros miembros del equipo o en proyectos open source, podría ser bastante tedioso tener que compartir esta carpeta, ya que suele ser bastante pesada.

Lo brillante de este sistema de dependencias es que solo necesitamos indicar en nuestro fichero de firma del proyecto las dependencias y su versión para que todos los que trabajen en el mismo proyecto tengan el mismo estado de la aplicación.

Es decir, en caso de trabajar en un proyecto con otros desarrolladores, al entrar en una empresa nueva o al compartir nuestro trabajo con el mundo, nuestro proyecto queda definido en el fichero `package.json` y todo el mundo tendrá las mismas versiones y dependencias al ejecutar el comando:

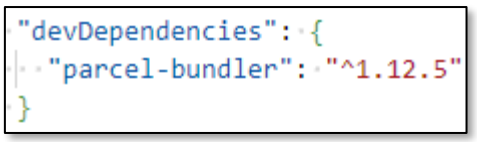
npm install

O su versión reducida:

npm i

Nota: Se espera que en las entregas de las practicas / exámenes no se incluya la carpeta **node_modules** puesto que la carga y descarga de los ejercicios aumenta de manera innecesaria.

Si revisamos nuestro `package.json`, veremos que ha aparecido un nuevo par de valores:



Hay dos tipos de dependencias: `**dependencies**` y `**devDependencies**`.

Las primeras son las que necesitaremos tanto en desarrollo como en producción, como podría ser alguna librería tipo `react` o `d3`.

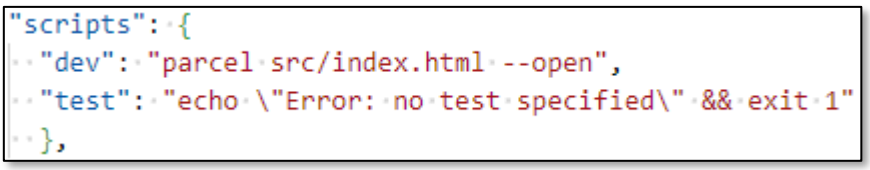
Las segundas son las dependencias de desarrollo, como puede ser la librería de testing `jest`.

No necesitaremos testing en nuestra versión de producción expuesta al público, pero sí lo necesitaremos en fase de desarrollo para poder testear nuestra aplicación.

5 NUESTRO PRIMER SCRIPT

Los scripts son comandos propios de cada proyecto que podemos definir para abstraer partes de la lógica asociada a los procesos de desarrollo. Vamos a crear un script para lanzar el live server para nuestra aplicación.

Vamos a añadir esta línea en nuestro fichero `**package.json**`:



Nota: El flag "--open" hace que el navegador se abra automáticamente cuando el servidor está listo. Eliminar este flag si queréis el comportamiento por defecto.

Para poder ejecutar el comando `dev`, debemos hacerlo de este modo desde la raíz del proyecto:

npm run dev

Nota: Si lo ejecutamos, veremos un error por pantalla puesto que no encuentra `index.html`.

Los beneficios inmediatos de esto son:

- No necesitamos que Parcel sea una dependencia global del sistema (ved la nota más abajo).
- Hemos abstraído el punto de entrada de la aplicación a un script del proyecto.
- Hemos generalizado la carga del proyecto a un comando `dev` de npm que es un estándar.

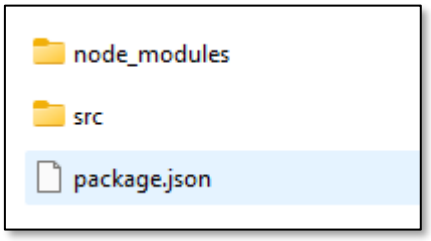
El uso de scripts es mucho más valioso cuando los comandos tienen muchas opciones o flags adicionales. Este es, por tanto, un ejemplo sencillo del uso que puede darse a los scripts.

Nota: De no ser así, los usuarios que accedieran al proyecto y no tuvieran Parcel como dependencia de sistema recibirían un error al ejecutar el script. Es mejor que todas las dependencias de un proyecto se incluyan en él.

5.1 PRIMEROS PASOS CON NUESTRA APLICACIÓN

Todo lo que hemos hecho ahora ha sido definir nuestro `tooling` y la base de nuestra aplicación. Vamos a añadir tres ficheros de html, css y js para entender cómo nos va a ayudar `parcel` en fase de desarrollo.

Creamos una nueva carpeta `src` en la raíz de nuestro proyecto:



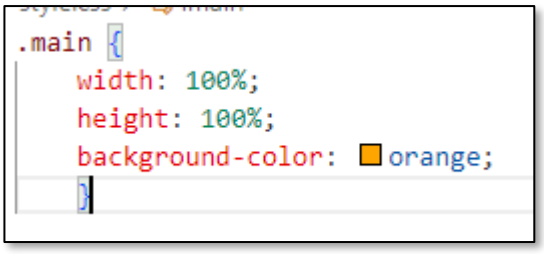
Abrimos nuestro editor y creamos en la carpeta recién creada el fichero `index.html` con este contenido:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="main">

    </div>

    <script src="eje.js"></script>
</body>
</html>
```

Crearemos también los ficheros style.css y eje.js

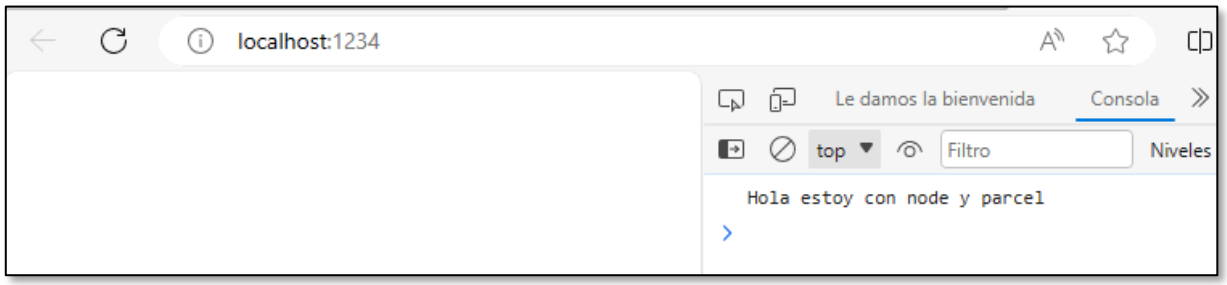
style.css	eje.js
	

Estos tres archivos son una de las representaciones más básicas de una aplicación de navegador con la estructura en html, los estilos en css y la interactividad proporcionada por JavaScript.

Volveremos a nuestro terminal y ejecutaremos:

npm run dev

Y se nos abrirá una ventana del navegador con nuestra página web.



Han sucedido procesos bastante sofisticados que han quedado escondidos a nuestros ojos para que podamos ver el resultado de la creación de nuestros archivos renderizados en el navegador.

Sin embargo, lo que puede resultarnos más útil para la realización de las prácticas es el hecho de que mientras no cerremos la consola en la que hemos ejecutado `parcel`, podremos modificar cualquiera de los ficheros y los cambios se recargarán automáticamente en el navegador.

Probad a añadir algún texto en el `index.html`, modificar alguno de los estilos o cambiar el texto del `console.log` y veréis que el navegador se autorrefresca y recarga los cambios.

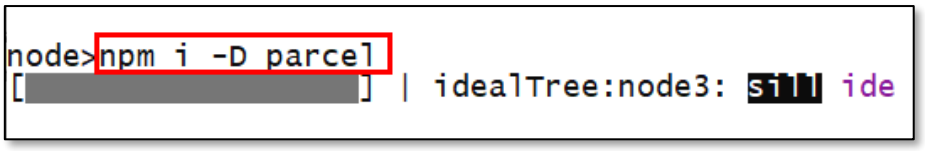
Nota: Es recomendable tener una distribución de pantalla partida en la que una mitad la ocupe el navegador con `localhost:1234` y la otra mitad vuestro editor para trabajar y ver los resultados de los cambios en la misma pantalla.

5.2 ÚLTIMOS RETOQUES A NUESTRA APLICACIÓN

Para que nuestra aplicación quede completa debemos terminar de configurar nuestro proyecto, para ello tendremos que instalar el paquete "**parcel**" y hacer unas pequeñas modificaciones en el fichero **package.json.**

Vamos a instalar **parcel** en nuestro proyecto:

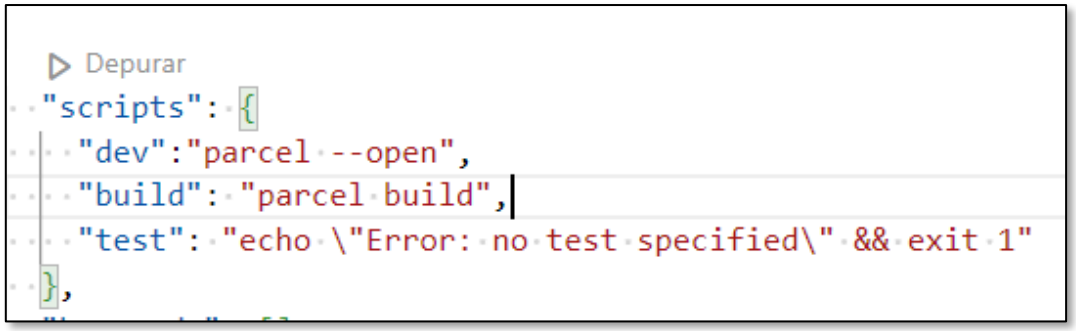
npm i -D parcel



Una vez instalado vamos a modificar el fichero **package.json** (de nuevo)

Debemos añadir un campo nuevo en este fichero llamado "**source**" donde vamos a indicar la pagina de inicio de nuestro proyecto, en nuestro caso "**src/index.html**"

Vamos a modificar dentro de "**scripts**" el script (valga la redundancia) "**dev**", además de añadir uno nuevo llamado "**build**" (que nos servirá para cuando subamos la aplicación a Netlify)



Ahora tenemos tres posibilidades: "**npm run dev"**, "**npm run build"** y "**npm run test"** (no implementada aun).

```
node>npm run dev

> node3@1.0.0 dev
> parcel --open

Server running at http://localhost:1234
√ Built in 738ms
```

```
node>npm run build

> node3@1.0.0 build
> parcel build

√ Built in 2.28s

dist\index.html             347 B    854ms
dist\index.c1f6178a.css     118 B    167ms
dist\index.11b6642b.js      111 B    283ms
```