Configurar un proyecto **Node.js** con **Parcel** y **jQuery**:

1. Instalar Node.js y npm

Si aún no tienes Node.js y npm instalados (vienen juntos), descárgalos desde nodejs.org y sigue las instrucciones de instalación. O bien:

https://nodejs.org/en/download/package-manager

Para Linux:

installs nvm (Node Version Manager)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash

download and install Node.js (you may need to restart the terminal)
nvm install 22

verifies the right Node.js version is in the environment
node -v # should print `v22.12.0`

verifies the right npm version is in the environment
npm -v # should print `10.9.0`

2. Crear un nuevo proyecto

a) **Si vas a subirlo a Github, crea primero un repositorio proyecto89 allí [v]Add, Add gitignore node y licencia Apache 2.0.**

Desde carpeta de "proyectos", por ejemplo:

git clone URL.git (tomada la URL de github botón verde) crea**:** proyectos**proyecto89. Entra en la carpeta proyecto89. Sigue paso 3.**

b) **Si no vas a hacer la parte del repositorio de Github en este momento:**

Crea una carpeta para tu proyecto y accede a ella desde la terminal:

mkdir mi-proyecto
cd mi-proyecto

code .

3. Inicializa un nuevo proyecto **Node.js**: (similar al arquetipo de Maven y la inicialización de git usando 'init')

```
npm init -y
```

`Esto genera el archivo package.json, que sirve para registrar dependencias, configurar scripts de ejecución (por ejemplo, start, test, etc.) y almacenar la información del proyecto (nombre, versión, descripción, etc.).`

4. Instalar Parcel y jQuery

Instala Parcel JS como una *dependencia de desarrollo* y jQuery como una *dependencia estándar*:

```
npm install --save jquery
npm install --save-dev parcel          (tarda un poco)
```

Viene a conseguirse el efecto de la extensión LiveServer en VSCode. Ver los cambios que genera el código en tiempo real en el navegador, pero con más enjundia.

Una posibilidad menos indicada sería:

npm install -g parcel

Esto realizaría una instalación global en el proyecto y no solo para la parte de desarrollo.

Tras instalar jQuery:

node_modules/: Esta carpeta contiene todas las dependencias de tu proyecto. Cuando ejecutas npm install jquery, npm crea la carpeta node_modules (si no existe) y descarga la librería jquery dentro de esta carpeta.

package.json: Se actualiza con la entrada para `jquery` en la sección `dependencies`, algo como esto:

```
json
Copiar código
"dependencies": {
  "jquery": "^3.6.0"
}
```

package-lock.json: Este archivo es generado automáticamente por npm para registrar las versiones exactas de las dependencias instaladas. Este archivo ayuda a garantizar que las mismas versiones se instalen en otros entornos de desarrollo, evitando discrepancias.

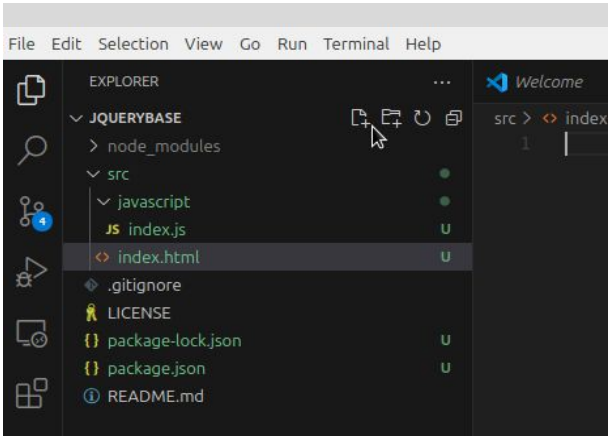
5. VSCode y la estructura de carpetas

Desde la carpeta del proyecto, abre una ventana de terminal y ejecuta VSCode:

code .

Asegúrate de tener una estructura básica como esta: (src/ debes crearla manualmente. Paso 5)

```
mi-proyecto/
├── node_modules/
├── package.json
├── src/
│   ├── javascript
│   │   └── index.js
│   └── index.html
└── .gitignore
```



6. Manualmente creamos src\javascript y los archivos index.js e index.html

Crea el archivo `index.html` dentro de la carpeta `src` con lo siguiente:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi Proyecto con Parcel y jQuery</title>
    <script src="../node_modules/jquery/dist/jquery.js"></script>

</head>
<body>
    <h1>Hola, mundo con Parcel y jQuery!</h1>
    <script src="./javascript/index.js"></script>

</body>
</html>
```

Añade a index.js un contenido con el que empezar, si quieres.

```
$(document).ready(function() {


});
console.log("¡jQuery funcionando!");
```

Añade a **.gitignore:**

```
node_modules/
dist/
```

Añade al archivo package.json:

Sección scripts debe quedar así:

```
"scripts": {
"server": "npx parcel src/index.html",
"build": "npx parcel build ./src/javascript/index.js",
"test": "echo \"Error: no test specified\" && exit 1"
},
```

Save All / Guardar todos los archivos.

package.json contendrá algo similar a esto:

```
{

"name": "jquerybase",
"version": "1.0.0",
"description": "Punto de partida proyecto nodejs - jquery - parcel js",
"main": "index.js",
"scripts": {
"server": "npx parcel src/index.html",
"build": "npx parcel build ./src/javascript/index.js",
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
"jquery": "^3.7.1"
},
"devDependencies": {
"parcel": "^2.13.2"
}
}
```

7. Abrir terminal en VSCode. (CTRL+SHIFT+`)

npm run server

.parcel-cache y **dist** aparecen tras haber levantado el servidor por primera vez: npm run server

Al levantar el servidor habilita el **http://localhost:1234/**
Los puertos para desarrollo en Javascript suelen ser: 1234, 4300 ó el puerto 3000.
Si abrimos el navegador en esa URL iremos viendo en tiempo real los cambios que realicemos en el código HTML y Javascript.

CTRL+SHIFT+I para abrir las herramientas de desarrollo y ver el mensaje de console.log que habíamos puesto en index.js

Crea el archivo `index.js` dentro de la carpeta `src` y usa jQuery en él:

```
import $ from 'jquery';       // Mejor no.
// No vamos a usar módulos en este curso, por lo que el import queda descartado.
// Emplear: CDN o bien lo siguiente (node_modules): (dentro del <head> de
index.html)

<script src="../node_modules/jquery/dist/jquery.js"></script>

Y esto en el index.js:

$(document).ready(function() {
    $('h1').text('¡Hola, mundo de jQuery con Parcel!');
});
```

7. Configurar el archivo `.gitignore`

Si estás utilizando Git, crea un archivo `.gitignore` para evitar que ciertos archivos se suban al repositorio:

```
node_modules/
dist/
```

8. Añadir un script de inicio en `package.json`

Abre el archivo `package.json` y añade un script para ejecutar Parcel:

```
"scripts": {
    "start": "parcel src/index.html"
}

En mi proyecto del curso, no hicimos una instalación 'global' de parcel. Eso nos
permite usar npx en lugar de npm. Es más corto el path y además garantiza la
compatibilidad de versiones.

"scripts": {
"server": "npx parcel src/index.html",
"build": "npx parcel build ./src/javascript/index.js",
"test": "echo \"Error: no test specified\" && exit 1"
}
```

9. Ejecutar el proyecto

En la terminal, ejecuta el siguiente comando **para iniciar el servidor de desarrollo de Parcel**:

```
npm start   # o mejor (si la instalación no ha sido 'global'):

npm run server   # en el caso del proyecto del curso.
```

Esto iniciará Parcel y abrirá el proyecto en tu navegador. Puedes hacer cambios en el archivo `index.js` o `index.html`, y Parcel recargará la página automáticamente.

localhost:1234

10. Verificar el funcionamiento

Abre tu navegador y verifica que el texto del encabezado `<h1>` cambie, indicando que jQuery está funcionando correctamente.

Diferencia entre `npx` y `npm run`:

- **`npx`**: Es útil cuando necesitas ejecutar un paquete **de manera temporal** (sin tenerlo como dependencia o cuando no está instalado globalmente). `npx` ejecuta un paquete sin necesidad de estar en el `package.json`, y es especialmente útil cuando no tienes configurado un script.
- **`npm run`**: Ejecuta un **script** que has definido en tu archivo `package.json`. Si en el script mencionas el nombre de un paquete (como `parcel`), **npm** buscará automáticamente ese ejecutable en `node_modules/.bin` y lo ejecutará. No necesitas usar `npx` en este caso, ya que **`npm run`** se encarga de ello.

Resumen:

1. **Inicializa el proyecto** con `npm init`.
2. **Instala Parcel y jQuery** con `npm install --save jquery` y `npm install --save-dev parcel`.
3. **Crea los archivos de entrada** (`index.html` y `index.js` en `src/`).
4. **Escribe el código en JS usando jQuery**.
5. **Configura el script de inicio** en `package.json`.
6. **Ejecuta el servidor con `npm start` o npm run server** (en el curso)

Con estos pasos, habrás configurado correctamente un proyecto de Node.js utilizando Parcel y jQuery.

En clase hemos usado esta configuración con estos scripts y dependencias:

package.json

```
{
"name": "proyecto14",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"server": "npx parcel src/index.html",
"build": "npx parcel build ./src/javascript/index.js",
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "Apache-2.0",
"dependencies": {
"jquery": "^3.7.1"
},
"devDependencies": {
"parcel": "^2.13.2"
}
}
```

El uso de **`npx`** en lugar de **`npm`** en tu script de servidor permite ejecutar **Parcel** sin necesidad de instalarlo globalmente o especificar su ruta dentro de `node_modules/.bin`. Esto simplifica la ejecución y asegura que siempre se use la versión correcta del paquete instalada en tu proyecto.

Sí, **Parcel JS** es una herramienta de empaquetado de aplicaciones web que facilita la construcción y el empaquetado de archivos de JavaScript (y otros recursos como HTML, CSS, imágenes, etc.) para que puedan ser utilizados en el navegador. A continuación, te explico en detalle por qué y cómo funciona:

¿Por qué Parcel (o cualquier empaquetador) es necesario para el navegador?

El navegador no puede interpretar directamente ciertos tipos de código de JavaScript o archivos modernos sin primero procesarlos y empaquetarlos de manera adecuada. Algunas razones por las que **Parcel** y otras herramientas de construcción (como Webpack o Rollup) son útiles incluyen:

1. **Compatibilidad con los módulos ES6 (ESModules)**:
    - En JavaScript moderno, es común usar **módulos ES6** (es decir, usar `import` y `export` en el código).
    - Los navegadores más antiguos no pueden entender módulos de ES6 de manera nativa, por lo que **Parcel** convierte estos módulos en un formato que el navegador puede entender. Parcel también se asegura de que los módulos se resuelvan correctamente, incluso si tienes dependencias externas.
2. **Transpilar código moderno (como JSX o TypeScript)**:
    - Si usas tecnologías como **React** (que usa JSX) o **TypeScript**, Parcel se encarga de **transpilar** ese código a JavaScript estándar que el navegador puede ejecutar. El navegador solo entiende JavaScript "puro", por lo que cualquier sintaxis que no sea compatible necesita ser transformada.
3. **Minificación y optimización**:
    - Parcel también optimiza tu código, reduciendo su tamaño mediante técnicas como la **minificación** y el **tree-shaking** (eliminación de código muerto). Esto mejora el rendimiento en el navegador, ya que se carga menos código y el sitio se vuelve más rápido.
4. **Embalaje de dependencias**:
    - Si tu proyecto depende de librerías externas (como jQuery, React, etc.), Parcel empaqueta esas dependencias en un único archivo o en archivos optimizados para ser cargados más eficientemente. Esto es especialmente importante para evitar múltiples peticiones HTTP innecesarias en el navegador.
5. **Hacer más fácil el desarrollo**:
    - Parcel proporciona un **servidor de desarrollo** que te permite trabajar de manera más rápida. Cada vez que haces cambios en tu código, Parcel recompila solo los archivos modificados y actualiza el navegador en tiempo real (con **Hot Module Replacement**, HMR). Esto mejora la experiencia de desarrollo.

¿Parcel es "imprescindible" para ver un proyecto en el navegador?

No es estrictamente "imprescindible", pero sí muy **útil y recomendable** para muchos proyectos por las siguientes razones:

- **Código moderno**: Si estás usando características de JavaScript modernas (como ES6+), Parcel facilita la compatibilidad con los navegadores actuales y antiguos.
- **Dependencias externas**: Si utilizas bibliotecas como **jQuery**, **React**, **Vue**, etc., Parcel se encarga de gestionarlas y empaquetarlas.
- **Optimización**: Parcel ayuda a optimizar el tamaño de los archivos, lo cual es esencial para proyectos grandes.

Sin embargo, si tu proyecto es muy sencillo y no estás utilizando características avanzadas de JavaScript o no dependes de librerías externas, podrías hacerlo sin un empaquetador. Solo necesitarías incluir los archivos de JavaScript y HTML directamente en tu proyecto, pero para proyectos más complejos, el uso de Parcel o un empaquetador similar es muy recomendable.

Resumen:

Parcel es una herramienta útil (pero no estrictamente imprescindible) que facilita la construcción, empaquetado, y optimización del código para ser ejecutado en el navegador. Empaquetar el código es necesario porque los navegadores necesitan archivos de JavaScript procesados correctamente, compatibles con el entorno de ejecución, y optimizados para rendimiento. Sin un empaquetador como Parcel, tendrías que gestionar manualmente estos procesos, lo cual sería mucho más complicado y propenso a errores.