



Clase 14. Programación Backend

Webpack: Module Bundler



OBJETIVOS DE LA CLASE

- Comprender el concepto de bundler.
- Instalación y uso de Webpack mediante Node.js.
- Integración de Webpack y Typescript en un proyecto Node.js.

CRONOGRAMA DEL CURSO

Clase 13



**Node.js como
herramienta de
desarrollo**

Clase 14



**Webpack: Module
Bundler**

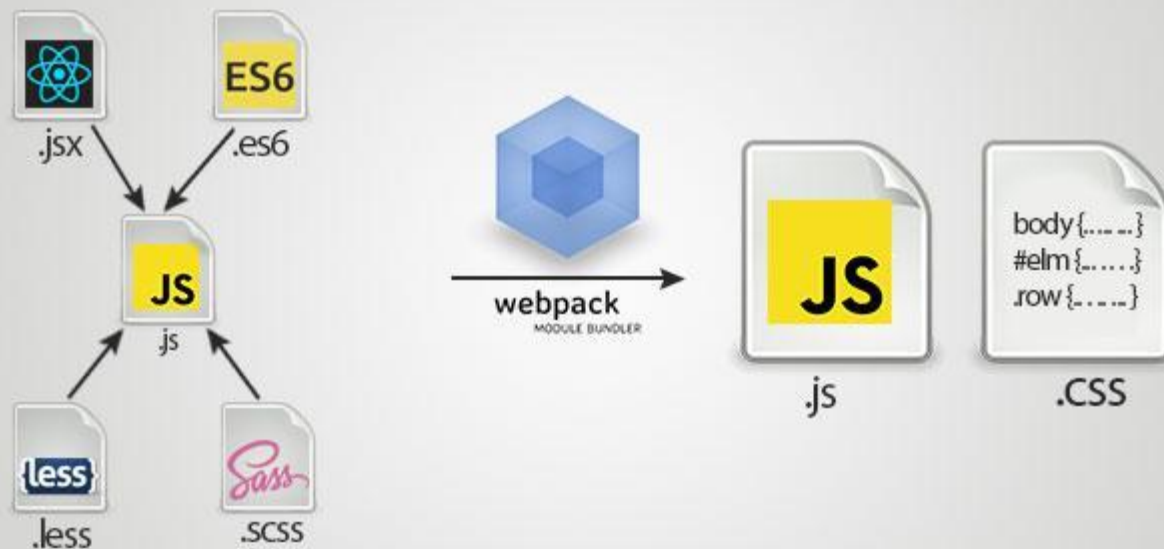


1º ENTREGA PROYECTO FINAL

Clase 15



Webpack: Module Bundler

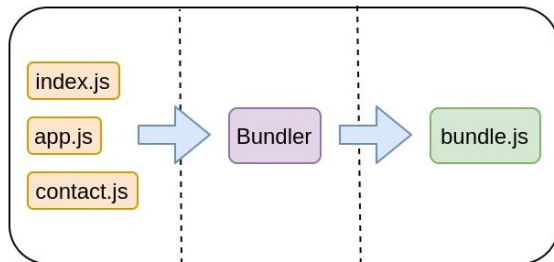




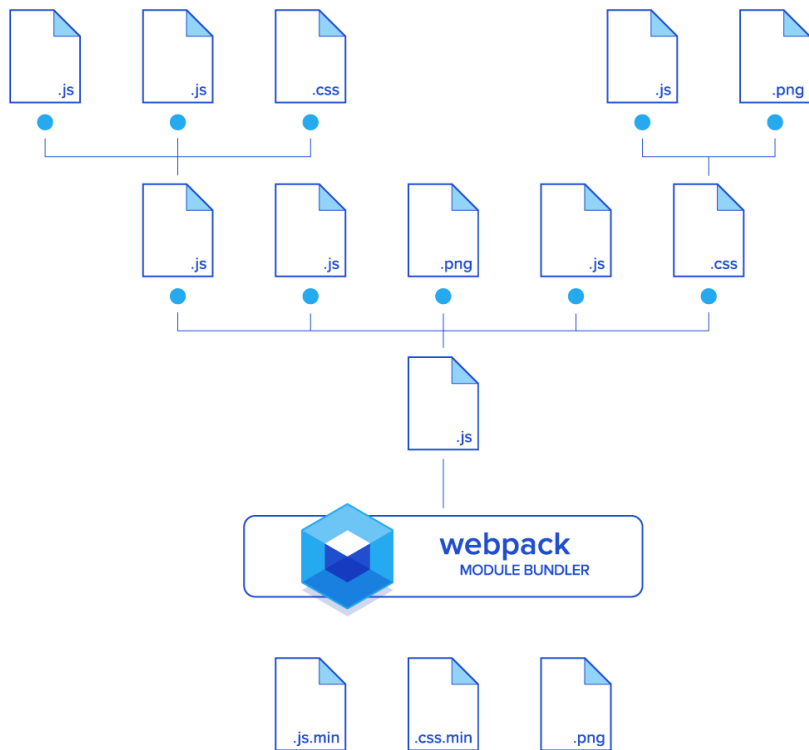
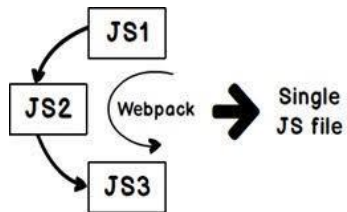
¿Qué es Webpack?



- **Webpack** es un **empaquetador de módulos** (*module bundler*), que genera un archivo único con todos los módulos que necesita la aplicación para funcionar.
- Permite encapsular todos los archivos JavaScript en un único archivo, por ejemplo *bundle.js*
- **Webpack** se ha convertido en una herramienta de build muy versátil.



Esquemas



webpack



Front-end bundler

```
// cow.js
export function cow() {
  console.log("Moo!");
}
```

```
// main.js
import { cow } from './cow.js';

cow();
```

```
function cow() {
  console.log("Moo!");
};

cow();
```

CODER HOUSE

Con Webpack vamos a poder...

- Generar solo aquellos fragmentos de JS que realmente necesita cada página (haciendo más rápida su carga).
- Disponer de varios loaders para importar y empaquetar también otros recursos (CSS, templates, ...) así como otros lenguajes (ES6 con Babel, TypeScript, SaSS, etc).
- Utilizar plugins que permiten hacer otras tareas importantes, como por ejemplo minificar y ofuscar el código.

Webpack y Node.js





Webpack y Node.js

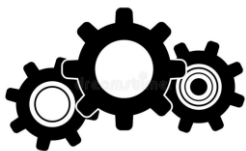


Existen varias formas de utilizar Webpack. Trabajaremos con la versión en línea de comandos (CLI) que realiza una empaquetación directa:

1. Creamos un proyecto de Node.js con **npm init -y**
2. Instalamos Webpack y Webpack CLI

npm install webpack webpack-cli

De los dos módulos instalados, el primero es el propio webpack y el segundo es la dependencia para usar webpack desde la consola de comandos (Command Line Interface).



Empaquetando módulos



A modo de ejemplo, vamos a generar tres archivos con contenido javascript (a1, a2, a3) y los vamos a empaquetar en un sólo archivo de salida ejecutando un comando definido en el *package.json* del proyecto:

```
"scripts": {  
  "build": "webpack ./a1.js ./a2.js ./a3.js"  
},
```

Por defecto, esto creará una carpeta *dist* con un archivo *main.js* dentro, que contiene la versión empaquetada de todos los archivos especificados.

En caso de no especificar, buscará un archivo *index.js* dentro de una carpeta *src* por defecto, e incluirá en forma recursiva todas las dependencias de ese archivo y de sus dependencias.

package.json

```
{
  "name": "ejemplo-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "./dist/main.js",
  "scripts": {
    "build": "webpack ./a1.js ./a2.js ./a3.js --mode=production",
    "dev": "webpack ./a1.js ./a2.js ./a3.js -w --mode=development",
    "start": "node ."
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.53.0",
    "webpack-cli": "^4.8.0"
  }
}
```

El modo modo desarrollo o producción define si el código generado tendrá formato de lectura amigable y comentarios, o si estará minificado, respectivamente.

Aclaración para nuevas versiones de node:

Webpack utiliza una versión de una librería interna de node que fue deprecada en Node v17. Para quienes quieran/deban usar esta última versión, pueden agregar una configuración del entorno de ejecución previa al lanzamiento del programa, cargando la siguiente variable de entorno:

(para linux/mac):






```
NODE_OPTIONS=--openssl-legacy-provider npm start
```

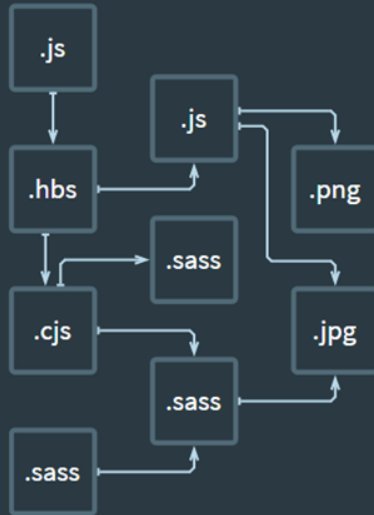
(o para windows cmd):

```
set NODE_OPTIONS=--openssl-legacy-provider && npm start
```

Webpack: Web oficial <https://webpack.js.org/>

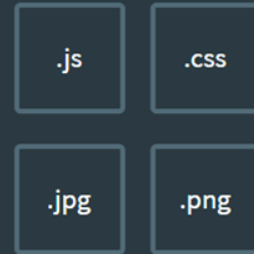
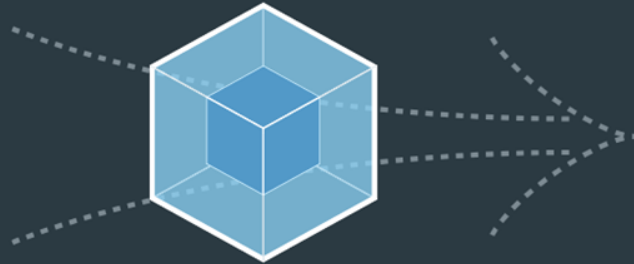


DOCUMENTATION CONtribute VOTE BLOG      



MODULES WITH DEPENDENCIES

bundle your scripts



STATIC ASSETS

Webpack: Documentación

<https://webpack.js.org/concepts/>

[DOCUMENTATION](#)[CONTRIBUTE](#)[VOTE](#)[BLOG](#)[API](#)[CONCEPTS](#)[CONFIGURATION](#)[GUIDES](#)[LOADERS](#)[MIGRATE](#)[PLUGINS](#)**webpack v5.21.2**[Print Section](#)

▼ Concepts

- Entry
- Output
- Loaders
- Plugins
- Mode
- Browser Compatibility
- Environment

- › Entry Points
- › Output
- › Loaders
- › Plugins
- › Configuration
- › Modules
- › Module Resolution
- › Module Federation
- › Dependency Graph
- › Targets

[EDIT DOCUMENT](#) | [Print Document](#)

Concepts

At its core, **webpack** is a *static module bundler* for modern JavaScript applications. When webpack processes your application, it internally builds a [dependency graph](#) which maps every module your project needs and generates one or more *bundles*.

Tip

Learn more about JavaScript modules and webpack modules [here](#).

Since version 4.0.0, **webpack does not require a configuration file** to bundle your project. Nevertheless, it is [incredibly configurable](#) to better fit your needs.

To get started you only need to understand its **Core Concepts**:

- [Entry](#)
- [Output](#)
- [Loaders](#)
- [Plugins](#)
- [Mode](#)
- [Browser Compatibility](#)

CODER HOUSE



MENSAJERÍA CON WEBPACK

Tiempo: 10 minutos



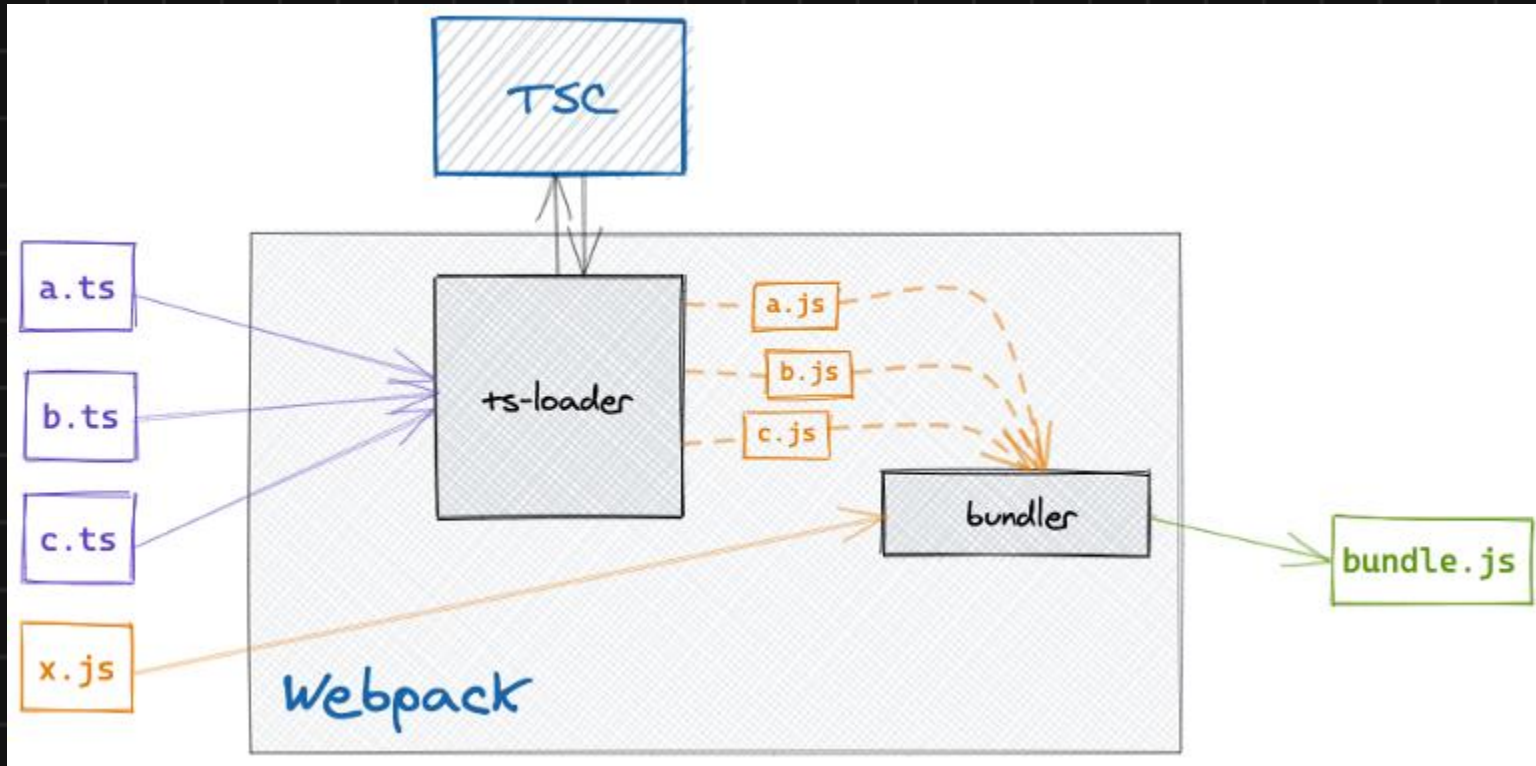
- Crear tres archivos javascript que contengan una variable por cada archivo llamadas *mensaje1*, *mensaje2* y *mensaje3* respectivamente.
- En cada archivo, cargar la variable con un frase y representarla en consola luego de:
 - Un segundo para el caso del mensaje 1.
 - Dos segundos para el mensaje 2.
 - Tres segundos para el mensaje 3.
- Crear un proyecto que permita utilizar webpack como dependencia de desarrollo para empaquetar los tres archivos en uno sólo.
- Escribir el script correspondiente para ejecutar el proceso automático, generando la versión de producción del proyecto.

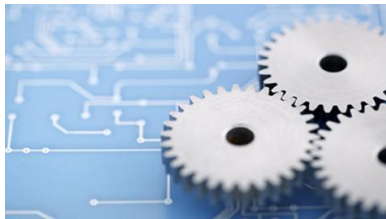


BREAK

¡5/10 MINUTOS Y VOLVEMOS!

Proyecto Webpack + Typescript





Creación del proyecto: pasos



Desarrollaremos un proyecto en el cual integraremos un servidor en node.js con **Webpack** y **Typescript** vinculados, que permitirán importar nuestros módulos *CommonJS* y *ES Modules*.

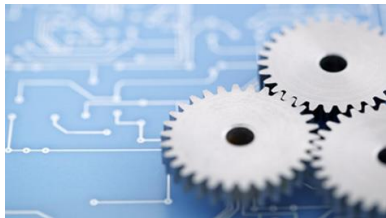
Vamos a seguir esta serie de pasos para crear el proyecto desde cero

1. Generamos la carpeta de proyecto
2. Inicializamos un proyecto de node con **npm init -y**
3. Dentro del proyecto creamos un carpeta **src** con un archivo **index.ts**.
4. Instalamos las dependencias de desarrollo:

npm i -D typescript ts-loader webpack webpack-cli

webpack node externals

CODER HOUSE



Creación del proyecto: pasos



5. Instalamos las dependencias del proyecto:
npm i express @types/express
5. Creamos el archivo *tsconfig.json* (configuración del transpilador typescript) con el comando ***./node_modules/.bin/tsc --init***
6. Modificamos *tsconfig.json* dejando la clave "***noImplicitAny***" en ***false*** (deshabilita la generación de errores en expresiones y declaraciones con cualquier tipo implícito)
7. Creamos el archivo ***webpack.config.js*** y le agregamos el siguiente contenido:

webpack.config.js

```
const path = require('path');
const nodeExternals = require('webpack-node-externals');

module.exports = {
  mode: 'production',
  entry: './src/index.ts',
  target: "node",
  externals: [nodeExternals()],

  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'main.js',
  },
  resolve: {
    extensions: ['.ts', '.js'],
  },
  module: {
    rules: [
      {
        test: /\.tsx?/,
        use: 'ts-loader',
        exclude: /node_modules/
      }
    ]
  }
}
```

Propiedades que podemos configurar

mode: para el modo de trabajo (*development* ó *production*)

entry: para definir el punto de entrada de nuestro código.

externals: permite el correcto funcionamiento con algunas librerías externas (en este caso, express)

output: para definir el punto de salida.

resolve: configura cómo se resuelven los módulos

module: sirve para aclararle a Webpack cómo debe procesar los loaders que queramos usar para un proyecto.

Aclaraciones acerca de la configuración de webpack

- Los **loaders** son transformaciones que se aplican en el código fuente de nuestras aplicaciones. Existen decenas de ellos, para usar cantidad de tecnologías y transformar código de preprocesadores, código HTML, Javascript, etc. Son como una especie de tareas que Webpack se encargará de realizar sobre nuestro código, cada una especializada en algo en concreto.
- **ts-loader** es un TypeScript loader para webpack.
- Mediante las **rules** definidas dentro de la entrada **module**, podemos establecer a qué archivos se aplican los loaders que sean necesarios.

9. En el *package.json* agregamos lo marcado en rojo

```
{
  "name": "ejercicio2",
  "version": "1.0.0",
  "description": "",
  "main": "dist/main.js",
  "scripts": {
    "build": "webpack",
    "start": "node ."
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "ts-loader": "^8.0.17",
    "typescript": "^4.1.5",
    "webpack": "^5.53.0",
    "webpack-cli": "^4.8.0",
    "webpack-node-externals": "^3.0.0"
  },
  "dependencies": {
    "@types/express": "^4.17.11",
    "express": "^4.17.1"
  }
}
```

10. Escribimos el siguiente código en src/index.ts :

```
import express from "express";
import { getTime } from "../lib/utils";
import Persona from "../Persona";

const p: Persona = new Persona("Coder", "House");

const app = express();

app.get("/", (req, res) => {
  res.send({
    time: getTime(),
    name: p.getFullName(),
  });
});

const PORT = 8080;
app.listen(PORT, () => {
  console.log(`conectado al puerto: ${PORT}`);
});
```


11.Creamos la clase persona en src/Persona.ts

```
export default class Persona {  
  private fname: string;  
  private lname: string;  
  
  constructor(fname: string, lname: string) {  
    this.fname = fname;  
    this.lname = lname;  
  }  
  
  getFullName(): string {  
    return `${this.fname} ${this.lname}`;  
  }  
}
```

12. Por último, la librería *utils* en `src/lib/utils.ts`

```
export const getTime = () => {  
  return {  
    fyh: new Date().toLocaleString(),  
    timestamp: Date.now(),  
  };  
};
```

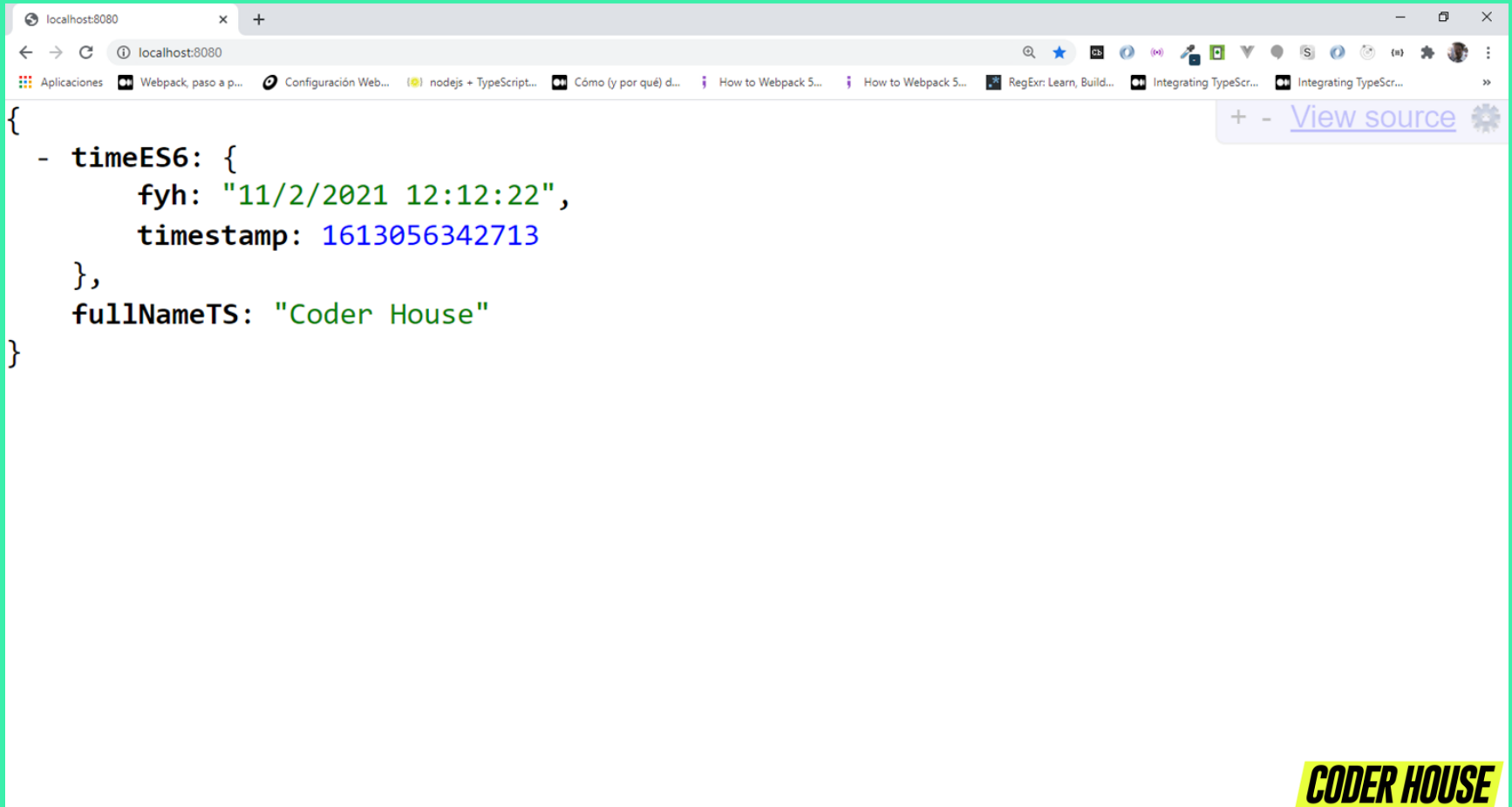


Puesta en marcha del proyecto



- Primero ejecutamos ***npm run build*** .
- Luego, ejecutamos ***npm start*** que lanzará la aplicación disponible en la carpeta **dist**.

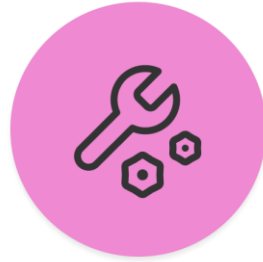
Proyecto completo : salida



The screenshot shows a web browser window with the address bar set to `localhost:8080`. The browser's developer tools are open, displaying a JSON object. The JSON structure is as follows:

```
{  
  - timeES6: {  
    fyh: "11/2/2021 12:12:22",  
    timestamp: 1613056342713  
  },  
  fullNameTS: "Coder House"  
}
```

In the bottom right corner of the browser window, there is a logo for "CODER HOUSE" in a stylized, bold, black font with a yellow outline.

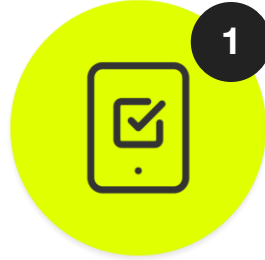


PERÍMETRO Y SUPERFICIE

Tiempo: 15 minutos



- Crear un proyecto basado en Webpack y Typescript que implemente un servidor node express cuyo punto de entrada será *server.ts*.
- Se implementará una clase llamada *Perimetro* que contenga tres métodos estáticos para calcular el perímetro de un cuadrado, un rectángulo y un círculo. Esta clase se guardará en un archivo llamado *perimetro.ts*
- En otro archivo llamado *superficie.ts* se implementará una clase llamada *Superficie* que contenga tres métodos estáticos para calcular la superficie de las mismas tres figuras.
- Los dos módulos se importarán en *server.js*.
- Realizar los endpoints *get* que permitan recibir las peticiones de cálculo con los parámetros correspondientes. La respuesta será en formato objeto y representará el tipo de cálculo, la figura, los parámetros de entrada y el resultado.
- Implementar el tipado en todas las clases y funciones del servidor.
- Probar con el navegador ó cliente http todas las posibles variantes.



PRIMERA ENTREGA DEL PROYECTO FINAL

Deberás entregar el avance de tu aplicación eCommerce Backend correspondiente a la primera entrega de tu proyecto final.

PRIMERA ENTREGA DEL PROYECTO FINAL

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Proyecto
Final



>>Consigna: Deberás entregar el estado de avance de tu aplicación eCommerce Backend, que implemente un servidor de aplicación basado en la plataforma Node.js y el módulo express. El servidor implementará dos conjuntos de rutas agrupadas en routers, uno con la url base '/productos' y el otro con '/carrito'. El puerto de escucha será el 8080 para desarrollo y process.env.PORT para producción en glitch.com

>>Aspectos a incluir en el entregable:

1. El **router base '/api/productos'** implementará cuatro funcionalidades:
 - a. GET: '/:id?' - Me permite listar todos los productos disponibles ó un producto por su id (disponible para usuarios y administradores)
 - b. POST: '/' - Para incorporar productos al listado (disponible para administradores)
 - c. PUT: '/:id' - Actualiza un producto por su id (disponible para administradores)
 - d. DELETE: '/:id' - Borra un producto por su id (disponible para administradores)

PRIMERA ENTREGA DEL PROYECTO FINAL

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Proyecto
Final



2. El **router base '/api/carrito'** implementará tres rutas disponibles para usuarios y administradores:
 - a. POST: '/' - Crea un carrito y devuelve su id.
 - b. DELETE:('/:id)' - Vacía un carrito y lo elimina.
 - c. GET:('/:id/productos)' - Me permite listar todos los productos guardados en el carrito
 - d. POST:('/:id/productos)' - Para incorporar productos al carrito por su id de producto
 - e. DELETE:('/:id/productos/:id_prod)' - Eliminar un producto del carrito por su id de carrito y de producto
3. Crear una variable booleana administrador, cuyo valor configuraremos más adelante con el sistema de login. Según su valor (true ó false) me permitirá alcanzar o no las rutas indicadas. En el caso de recibir un request a una ruta no permitida por el perfil, devolver un objeto de error.
Ejemplo: { error : -1, descripcion: ruta 'x' método 'y' no autorizada }

PRIMERA ENTREGA DEL PROYECTO FINAL

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Proyecto
Final



5. Un **producto** dispondrá de los siguientes campos: id, timestamp, nombre, descripcion, código, foto (url), precio, stock.
6. El **carrito de compras** tendrá la siguiente estructura:
id, timestamp(carrito), productos: { id, timestamp(producto), nombre, descripcion, código, foto (url), precio, stock }
5. El timestamp puede implementarse con Date.now()
6. Realizar la persistencia de productos y del carrito de compras en el filesystem.

PRIMERA ENTREGA DEL PROYECTO FINAL

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Proyecto
Final



>>A tener en cuenta:

1. Para realizar la **prueba de funcionalidad** hay dos opciones:
 - a. Probar con postman cada uno de los endpoints (productos y carrito) y su operación en conjunto.
 - b. Realizar una aplicación frontend sencilla, utilizando HTML/CSS/JS ó algún framework de preferencia, que represente el listado de productos en forma de cards. En cada card figuran los datos del producto, que, en el caso de ser administradores, podremos editar su información. Para este último caso incorporar los botones *actualizar* y *eliminar*. También tendremos un formulario de ingreso de productos nuevos con los campos correspondientes y un botón *enviar*. Asimismo, construir la vista del carrito donde se podrán ver los productos agregados e incorporar productos a comprar por su id de producto. Esta aplicación de frontend debe enviar los requests *get*, *post*, *put* y *delete* al servidor utilizando fetch y debe estar ofrecida en su espacio público.

CODER HOUSE

PRIMERA ENTREGA DEL PROYECTO FINAL

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Proyecto
Final



2. En todos los casos, el diálogo entre el frontend y el backend debe ser en formato JSON. El servidor no debe generar ninguna vista.
3. En el caso de requerir una ruta no implementada en el servidor, este debe contestar un objeto de error: ej { error : -2, descripcion: ruta 'x' método 'y' no implementada }
4. La estructura de programación será ECMAScript, separada tres en módulos básicos (router, lógica de negocio/api y persistencia). Más adelante implementaremos el desarrollo en capas. Utilizar preferentemente clases, constructores de variables let y const y arrow function.
5. Realizar la prueba de funcionalidad completa en el ámbito local (puerto 8080) y en glitch.com

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Concepto de bundler.
- Instalación y uso de Webpack mediante Node.js.
- Integración de Webpack y Typescript en un proyecto Node.js.



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN