

---

# UT5: Peticiones HTTP - Postman, NPM

Fréderic Sánchez García



## UT5.1.1 - Node

---

**Node.js** fue creado por los desarrolladores originales de **JavaScript**. Lo transformaron de algo que solo podía ejecutarse en el navegador en algo que se **podría ejecutar en los ordenadores como si de aplicaciones independientes se tratara**. Gracias a Node.js se puede ir un paso más allá en la programación con JavaScript no solo creando sitios web interactivos, sino teniendo la capacidad de hacer cosas que otros lenguajes de secuencia de comandos como Python pueden crear.

En nuestro caso **vamos a utilizar node para crear un servidor web** que atenderá peticiones HTTP permitiendo **crear, modificar, borrar y consultar datos**.

# UT5.1.2 - NVM

---

NVM es un gestor de versiones de **node** (**permite ejecutar javascript**), nos va permitir tener varias versiones de **node** en la misma máquina y elegir cual queremos utilizar.

La instalación la podemos hacer desde el siguiente [link](#).

Desde el terminal podemos ejecutar los siguientes comandos para usar nvm.

nvm list	<b>Lista</b> las versiones de node
nvm install VERSION	<b>Instala</b> la versión de node indicada
nvm install latest	<b>Instala</b> la última versión de node
nvm use VERSION	<b>Usa</b> la versión de node indicada
nvm --version	<b>Muestra</b> la versión del paquete nvm

Instalamos nvm y node

# UT5.1.3 - NPM (Node Package Manager)

---



npm es un gestor de paquetes (con sus dependencias) de node, también permite realizar tareas de automatización con su CLI (command line interface). **Se instala cuando instalamos node.**

Aunque lo veremos en siguientes temas, analizamos cómo podríamos instalar los paquetes typescript y angular-cli con npm (no los vamos a instalar para este tema):

- **typescript** es un lenguaje escrito en javascript. No es interpretado por los navegadores, lo que hace es dar más funcionalidad al lenguaje javascript y facilitar el desarrollo de aplicaciones que luego se traducen a javascript.
- **angular-cli** es una interfaz de uso de Angular (es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página) mediante línea de comando.

<b>npm install -g typescript</b>	<b>Instala</b> typescript en todo el equipo (-g = global)
<b>npm install typescript</b>	<b>Instala</b> typescript de forma local (en la carpeta)
<b>npm install -g @angular/cli</b>	<b>Instala</b> angular-cli

## UT5.1.3 - NPM - Comandos de ayuda

---

**NPM** se ha instalado como un comando de manera global en nuestro sistema. Eso significa que desde cualquier ruta podemos ejecutar npm en la consola/terminal.

Existen una serie de comando en npm por los que podemos consultar la ayuda:

- **npm -h:** Nos devuelve la ayuda y una lista de los comandos npm típicos.
- **npm <comando> -h:** Ayuda rápida sobre cómo se utiliza un determinado comando.
- **npm help <comando>:** Ayuda más extensa sobre un comando, o bien se abre la página del manual (en Linux) o bien se abre un navegador con la info.
- **npm help-search [palabras] :** Devuelve una lista con los temas de ayuda que contienen esas palabras.

# UT5.1.4 - API REST con Node + Express:

Vamos a crear nuestra propia API REST utilizando **Node** y **Express** (Framework backend de node), para ello vamos a seguir los pasos creados en la siguiente web.

<https://fsangar.github.io/ServidorJSONNodeExpress/>

The screenshot shows a dark-themed web application interface. On the left, there's a sidebar with buttons for 'Página inicial', 'Configuración avanzada' (which is highlighted in red), and 'Descargar el proyecto completo'. Below these are credits: 'This project is maintained by fsangar' and 'Contenido elaborado por Frédéric Sánchez García'. The main content area has a light background and a title 'Configuración avanzada'. It contains instructions: 'En este punto vamos a configurar el servidor para que atienda sobre una estructura más avanzada de JSON'. Step 1: 'Modificamos el fichero destinos.json donde se almacenarán la inforamción del JSON'. A terminal window shows the command 'mkdir db/destinosAdvanced.json'. Below it, it says 'Creamos un array de objetos en JSON con los campos id, name, lat y long' and shows a JSON code snippet:

```
[  
  {  
    "id": 1,  
    "name": "Mérida",  
    "lat": 38.928903210758286,  
    "long": -6.341210412197735  
  },  
  {  
    "id": 2,
```

A yellow callout box on the right says 'Creamos los dos servidores'.

## UT5.2 - Métodos básicos de HTTP:

---

Las peticiones y respuestas de los servidores son enviadas utilizando el protocolo HTTP (Hypertext Transfer Protocol).

Cuando hacemos una petición HTTP, el navegador envía al servidor unas **cabeceras-headers** (como el userAgent, Content-Type, ...), el **método de petición HTTP** y los **datos** (si es necesario).

Los métodos de petición HTTP más utilizados empleando AJAX para acceder a recursos o servicios son:

- **GET** → Es una petición por un recurso, pero no va a modificar nada en el servidor (similar a un SELECT en SQL). Usando este método los datos están codificados en la URL.
- **POST** → Es la operación por defecto para insertar nueva información y guardarla en el servidor (como el INSERT en SQL). Los datos están dentro del mensaje HTTP para que viajen de forma segura (logins).
- **PUT** → Es la operación que actualiza información que ya existe en el servidor. Similar a UPDATE en SQL → La información que identifica el objeto a actualizar va por la URL y los datos van dentro del mensaje.
- **DELETE** → Operación que borra los datos existentes en el servidor. Equivalente a DELETE en SQL. La información del objeto a borrar va en la URL.

# UT5.2 - Métodos básicos de HTTP - Postman:

---

Para probar estas peticiones se ha creado un servicio web alojado en la dirección: **localhost:3000** que maneja tareas que se deben realizar. Cada tarea tiene los siguientes campos:

## TAREA:

- \_id: Es un identificador en forma de cadena que toma un valor por defecto.
- título: Es una cadena, obligatoria.
- descripcion: Es una cadena, también obligatoria.
- fecha\_creacion: Es un Date, por defecto tiene el valor actual de la fecha.
- estado : Es una cadena que sólo puede tener los valores:
  - pendiente: Si sabemos que tenemos una tarea, pero todavía no la hemos empezado. Por defecto.
  - haciendo: Si ya hemos empezado la tarea.
  - completada: Si ya la hemos realizado.

La lista de tareas se maneja desde el servidor. A nosotros como clientes lo que nos interesaría: pedir las tareas, añadir tareas, modificar tareas y eliminar tareas. Es decir las operaciones **CRUD**:

**Create-Read-Update-Delete** típicas de una API de tipo **REST**.

# UT5.2 - Métodos básicos de HTTP - Postman:

---

El servidor que maneja las tareas tiene 2 recursos disponibles que aceptan diferentes tipos de peticiones HTTP:

- **localhost:3000/tasks**: Permite manejar las tareas en general. Acepta dos tipos de mensajes HTTP:
  - **GET**: Si realizamos un get sobre **localhost:3000/tasks** se nos devolverá un archivo JSON con un array que contiene todas las tareas almacenadas en el servidor.
  - **POST**: Si hacemos un post sobre **localhost:3000/tasks** y enviamos en el cuerpo del mensaje un objeto JSON que tenga los atributos nombre, descripcion, y opcionalmente fecha\_creacion y estado. Entonces se crearía una tarea con dichas características
- **localhost:3000/tasks/[id]**: Permite operar sobre una tarea en concreto. Acepta mensajes HTTP:
  - **GET**: Devuelve la tarea correspondiente con dicho **id** como un JSON.
  - **PUT**: Modifica la tarea correspondiente con dicho **id**. La tarea actualizada se debe pasar en el cuerpo de la petición.
  - **DELETE**: Borra la tarea correspondiente con dicho **id**.

# UT5.2 - Métodos básicos de HTTP - Postman:

Para probar que todas estas peticiones funcionan vamos a utilizar Postman.

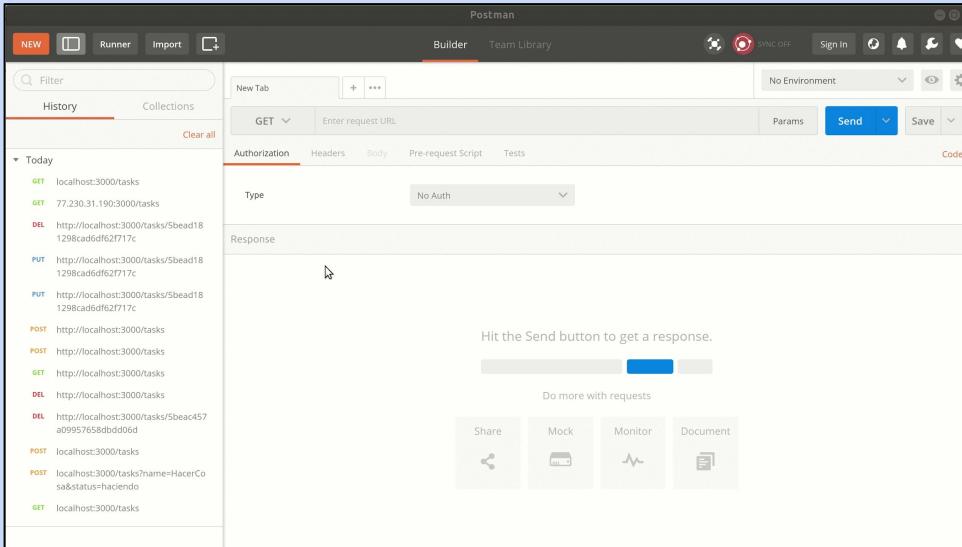
The screenshot shows the Postman application interface. At the top, there's a navigation bar with links for Home, Workspaces, API Network, Reports, and Explore. A search bar is also at the top. Below the navigation, the workspace title 'Probar clase 2021' is displayed, along with a 'New' button, an 'Import' button, and an 'Overview' button. To the right of the workspace title, it says 'No Environment'. On the left side, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main content area is titled 'Probar clase 2021' and contains a summary placeholder: 'Add summary to briefly explain what this workspace is all about...'. It also includes a 'Get started' section with links to Create a request, Create a collection, Create an API, Create an environment, and View More. The central part of the screen shows statistics: 2 Collections, 0 APIs, 0 Environments, 0 Mock Servers, and 0 Monitors. Below this, there's an 'Activity' section with a dropdown menu for User and Entity. The activity log shows several events from yesterday:

- JRedondo deleted the 'Peticiones Iniciales' collection at 12:12 PM
- JRedondo deleted the 'Peticiones Iniciales' collection at 12:12 PM
- JRedondo shared the 'Peticiones Iniciales' collection to this workspace at 12:11 PM
- JRedondo created the fork 'Peticiones Iniciales' from the 'Peticiones Iniciales' collection at 12:11 PM
- JRedondo edited the 'Peticiones de prueba' collection. View Changelog at 1:26 PM
- JRedondo shared the 'balldontlie' collection to this workspace at 5:22 PM

At the bottom, there are sections for 'Sharing' and 'Visibility', indicating that the workspace is 'Personal' and only you can access it.

# UT5 - Para entregar:

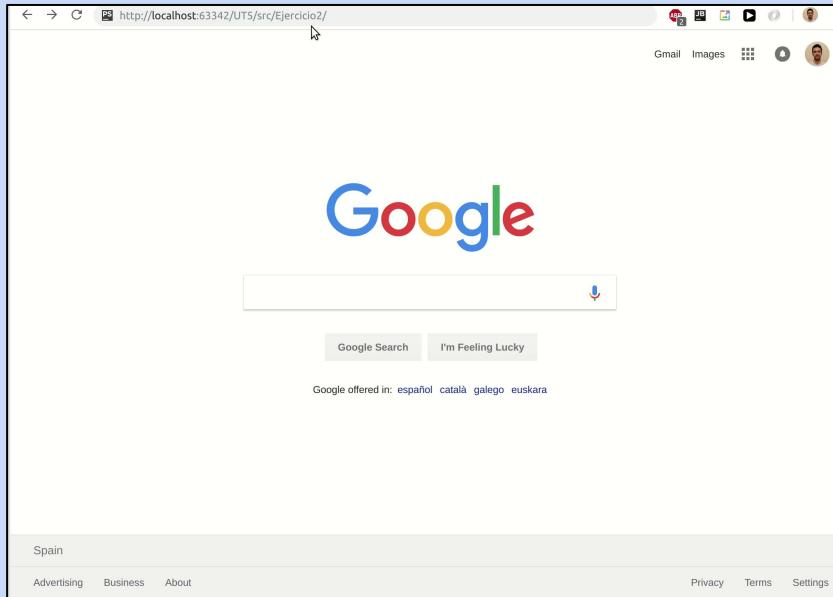
**Ejercicio 1:** Utiliza POSTMAN para crear, modificar, borrar y listar varias tareas. En esta tarea simplemente debes jugar con la API **localhost:3000** proporcionada. Presta especial atención en los errores que devuelve POSTMAN cuando no se pueden realizar las peticiones (404). Haz capturas de pantalla y crea un documento con todas las capturas de pantalla.



# UT5 - Para entregar:

---

**Ejercicio 2:** Crea una web que te permita consultar todas las tareas que hay en el servicio **localhost:3000/tasks**. Funcionaría de la misma manera que has realizado en Museos o cualquier otro dataset anteriormente.



# UT5 - Para entregar:

---

**Ejercicio 3:** Crea una web que te permita enviar una nueva tarea, para ello la web tendrá los campos suficientes para que el usuario pueda escribir la tarea y enviarla. Recuerda que esta vez tienes que realizar una petición POST (ojo que fetch por defecto hace get) y enviar dentro del cuerpo un objeto JSON con la tarea correspondiente. Revisa la siguiente pregunta de StackOverflow para ver cómo usar fetch con POST y enviar un objeto JSON: [PREGUNTA STACK OVERFLOW](#).

Ejemplo de como hacer un POST con fetch.

Es obligatorio chequear que la respuesta es correcta (de tipo 200) y que se ha podido crear la tarea.

Nueva tarea

Título	<input type="text" value="Título de la tarea"/>
Descripción	
<input type="text" value="Introduce la descripción"/>	
Estado:	<input type="text" value="Pendiente"/>
Fecha:	<input type="text" value="mm/dd/yyyy"/>
Hora:	<input type="text" value="--:-- --"/>
<input type="button" value="Enviar"/>	

# UT5 - Para entregar:

---

**Ejercicio 3:** Por defecto se checkean los campos para que si no hay título ni descripción no podamos enviar nada:

The screenshot shows a modal dialog titled "Nueva tarea" (New task). The form contains the following fields and their current values:

- Título:
- Descripción:
- Estado:
- Fecha:
- Hora:

A red asterisk (\*) is displayed next to the "Título" field, indicating it is required. A red error message "El campo Título es obligatorio." is shown above the "Título" input field. The number "1" is displayed at the bottom center of the modal, likely indicating the count of required fields that have not been filled.

# UT5 - Para entregar:

---

**Ejercicio 3:** Si la petición es correcta, se debería poder escribir los datos en el servidor. Se nos reenvía a la página de tareas:

Nueva tarea

Título: Terminar los apuntes de la UT5

Descripción:

Preparar los gifs para que se vean en ejecución el ejercicio-2 y el ejercicio-3.  
También prepararemos NPM para la siguiente semana.

Estado: Pendiente

Fecha: mm/dd/yyyy

Hora: 01:03 PM

Enviar

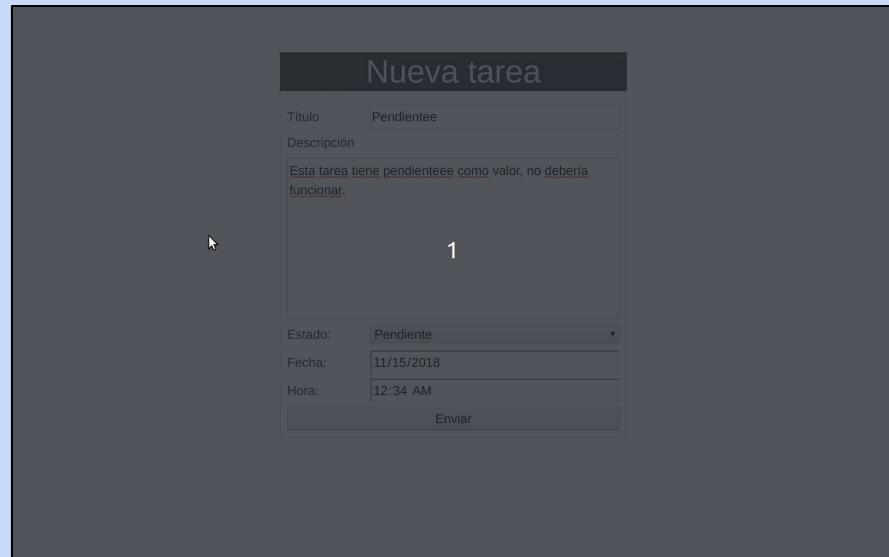
# UT5 - Para entregar:

---

**Ejercicio 3:** Comprobar errores (aunque por defecto los estamos evitando por el formulario):

Si por ejemplo el selector tuviese un estado incorrecto para el campo estado: "pendientee" ...

```
<textarea id="descripcion" class="w-100" placeholder="1<br><br>Tarea: <br><br><input type="text" value="Tarea: 1" /><br><br><label for="estado" class="w-25">Estado:</label><select id="estado" class="w-75"><option value="pendiente">Pendiente</option><option value="haciendo">Haciendo</option><option value="completada">Completada</option></select>
```



Nos devolvería un error mediante un alert.

# UT5 - Para entregar:

---

**Ejercicio 4:** Añadir un apartado en a web para que podamos modificar tareas. La interfaz es libre, cada uno puede realizarla como deseé.

Para modificar una tarea es necesario utilizar PUT.

Diseña una interfaz que solamente te deje modificar los posts ya creados.

**IMPORTANTE:** Para facilitar este ejercicio, sería muy recomendable estructurar las Tareas desde la clase “Tareas”.

Nueva tarea

Titulo: Pendienteee

Descripción:  
Esta tarea tiene pendienteee como valor, no debería funcionar.

Estado: Pendiente

Fecha: 11/15/2018

Hora: 12:34 AM

Enviar

Si no está contemplado en el servidor, introducelo en el servidor.

# UT5 - Para entregar:

---

**Ejercicio 5:** Añade un nuevo apartado a la web que permita eliminar Tareas.

## Tareas en el servidor

Jesús Probando de nuevo. Modificando la descripción  <span style="background-color: orange; border: 1px solid black; padding: 2px;">haciendo</span> 11/16/2018 -- 2:29:59 AM	Dar de comer a la mascota Pedir pienso para la mascota. Que sea royal canin.  <span style="background-color: green; border: 1px solid black; padding: 2px;">completada</span> 11/14/2018 -- 4:47:45 PM	Hacer un bocata Ya me va entrando hambre, tengo que hacer un bocata para pasar la media mañana sin que suene la tripa demasiado. Mejor uno vegetal  <span style="background-color: orange; border: 1px solid black; padding: 2px;">haciendo</span> 11/15/2018 -- 12:34:00 PM
Hacer la comida Macarrones con queso  <span style="background-color: red; border: 1px solid black; padding: 2px;">pendiente</span> 11/14/2018 -- 4:46:27 PM	Probando con otra tarea más Muy corta  <span style="background-color: green; border: 1px solid black; padding: 2px;">completada</span> 1/1/2000 -- 12:00:00 AM	Terminar los apuntes de la UT5 Preparar los gifs para que se vean en ejecución el ejercicio-2 y el ejercicio-3. También prepararemos NPM para la siguiente semana.  <span style="background-color: orange; border: 1px solid black; padding: 2px;">haciendo</span> 11/15/2018 -- 1:03:00 PM

# UT5 - Para entregar:

---

**Ejercicio 6:** Modifica toda tu interfaz para que todos los documentos HTML tengan una cabecera que nos permita seleccionar la acción a realizar con las Tareas.

The screenshot shows a web-based task management application. At the top, there's a navigation bar with icons for 'Tareas', 'Crear nueva tarea', 'Modificar Tarea', and 'Eliminar tarea'. Below the navigation, the title 'Tareas en el servidor' is displayed. The main content area lists five tasks in a grid format:

Tarea	Descripción	Estado	Última Actualización
Jesús	Probando de nuevo. Modificando la descripción	haciendo	11/16/2018 -- 2:29:59 AM
Dar de comer a la mascota	Pedir pienso para la mascota. Que sea royal canin.	completada	11/14/2018 -- 4:47:45 PM
Terminar los apuntes de la UT5	Preparar los gifs para que se vean en ejecución el ejercicio-2 y el ejercicio-3. También prepararemos NPM para la siguiente semana.	haciendo	11/15/2018 -- 1:03:00 PM
Hacer la comida	Macarrones con queso	pendiente	11/14/2018 -- 4:46:27 PM
Hacer un bocata	Ya me va entrando hambre, tengo que hacer un bocata para pasar la media mañana sin que suene la tripa demasiado. Mejor uno vegetal	haciendo	11/15/2018 -- 12:34:00 PM

Hay muchas maneras de llevar a cabo esta solución. Es responsabilidad tuya crear una solución que elimine totalmente el código duplicado de la cabecera en varios lugares.

Una vez has decidido e implementado una solución para que existan estas cabeceras... ¿Qué problemas puede tener tu solución?

## UT5.3.3 - NPM -Paquetes y módulos

---

**Sobre paquetes y módulos:** El registro/repositorio de npm contiene paquetes los cuales son también módulos de Node. Veamos cómo interactúan:

**LOS PAQUETES (packages):** Un **package** es un directorio que es descrito mediante un fichero **package.json**. Un **package** debe incluir package.json para que pueda ser publicado en el registro de npm.

Los packages pueden ser sin **scope** en cuyo caso no están recogidos dentro de ninguna organización (por ejemplo el paquete “lodash” (una librería de javascript )). Otros pueden estar dentro de una organización, por ejemplo “@koreez/phaser3-ninepatch”.

La página oficial es <https://www.npmjs.com/>

## UT5.3.3 - NPM -Paquetes y módulos

**Los formatos de los paquetes:** Cualquiera de los siguientes formatos puede ser un **package** de npm:

- a) Una carpeta que contiene un programa descrito con un fichero **package.json**.
- b) Una carpeta comprimida como formato tar que contiene a).
- c) Una URL que se resuelve a b).
- d) Una URL de git que cuando es clonada resulta en a).

**LOS MÓDULOS:** Un módulo es cualquier fichero o directorio dentro del directorio **node\_modules** que puede ser cargado por node.js con la función **require()**. Un módulo **debe ser alguna de las siguientes cosas:**

- Una carpeta con un **package.json** que contiene un campo “main”.
- Una carpeta con un fichero **index.js** en su interior.
- Un fichero JavaScript.

# UT5.3.3 - NPM -Paquetes y módulos

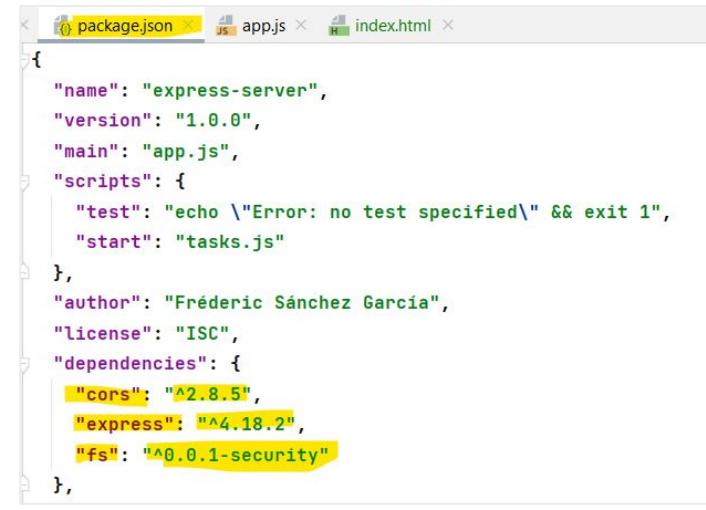
En el contexto de un programa de Node, el módulo son las dependencias que se cargan:

```
var moduloCargado = require('moduloQuePedimos')
```

Si echamos un vistazo al servidor que montamos en *API REST con Node + Express* vemos esto de forma clara.



```
/* @author Fréderic Sánchez García
*/
// Permite escribir en un fichero lo usaremos como base de datos para mantener los cambios
var fs = require("fs");
// Soluciona error CORS
const cors = require('cors');
// Creamos una instancia de express y le decimos que va a usar JSON
var express = require("express");
var app = express();
app.use(express.json());
// Evitar CORS
app.use(cors({
    origin: 'http://localhost:4200'
}));
```



```
{
  "name": "express-server",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "tasks.js"
  },
  "author": "Fréderic Sánchez García",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "fs": "^0.0.1-security"
  }
},
```

## UT5.3.4 - NPM - Inciso sobre require:

¿Qué es require? En **node.js** en el que todo el código está basado en javascript aparece el concepto de require. Podemos entenderlo de una manera similar a como funciona el **import** en otros lenguajes de programación. Nos permite cargar Objetos/Funciones/Valores que están descritas en otros **módulos**.

Para que esas otras Clases/Funciones/Valores puedan ser **requeridas** es necesario que sean **exportadas**.

Una de las principales diferencias entre import de otros lenguajes y require es la granularidad de lo que importamos. Mediante **exports** y **module.exports** podemos definir qué es lo que vamos a exportar para que pueda ser importado desde otro lado. Veamos un ejemplo:

fichero1.js

```
let invisible = function () {  
    console.log("invisible");  
};  
  
exports.message = "hi";  
  
exports.say = function () {  
    console.log(message);  
};
```

otroFichero.js

```
let variable = require('./ficher01.js');
```

variable vale  
ese objeto:

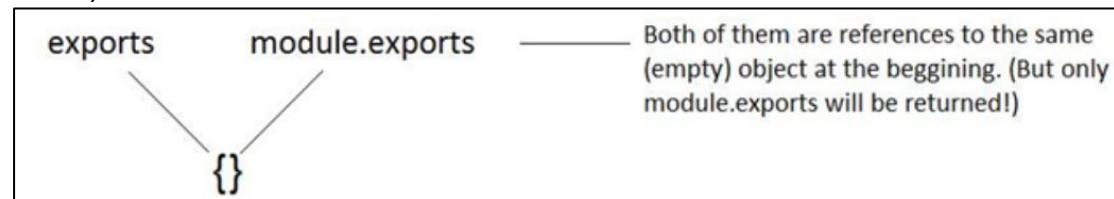
```
{  
    message: "hi",  
    say: [Function]  
}
```

**IMPORTANTE:**  
Require/export sólo  
funciona con **node.js**  
¡Desde el navegador no  
se puede utilizar!

## UT5.3.4 - NPM - Inciso sobre require:

Existen dos maneras de exportar una Objeto/Función con exports. Para entenderlo, tenemos que imaginar que al principio de cada fichero \*.js en el que exportemos existiera la siguiente declaración (que obviamente no hay que poner, es sólo ilustrativo):

```
var module = new Module(...);  
var exports = module.exports;
```



Entonces la diferencia entre uno y otro es:

- **exports:** Exporta cada uno de las Clases/Funciones/Valores que queramos, pero para exportarlo tenemos que hacer uso de **exports.<identificador> = Objeto/función/valor**. Es decir, modificamos los atributos del objeto que exportamos, pero no podemos modificar todo el objeto.
- **module.exports:** Puede funcionar de la misma manera que exports → **module.exports.<identificador> = Objeto/función/valor** pero a diferencia de exports, también se puede utilizar sin atributos, ¡Asignando directamente un objeto!. ESTO PROVOCA QUE CUANDO LO IMPORTEMOS DIRECTAMENTE ACCEDAMOS AL OBJETO/FUNCIÓN/VALOR.

## UT5.3.5 - NPM - Creación de packages:

Como hemos mencionado anteriormente, un package en npm es simplemente un directorio que tiene un fichero package.json con información sobre el propio proyecto. Vamos a inicializar un **package** mediante los pasos siguientes:

Abrimos una consola y nos dirigimos a una carpeta vacía donde vamos a comenzar nuestro proyecto.

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5$ mkdir UT5-NPM  
jesus@jesus-x470:~/Dropbox/DAW I/UT5$ cd UT5-NPM/
```

Para inicializar el package basta con ejecutar el comando:

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-NPM$ 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.
```

Para saber qué es lo que significa cada uno de los campos de package.json, puedes emplear el siguiente comando:

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-NPM$ npm help package.json
```

## UT5.3.5 - NPM - Creación de packages:

Tras finalizar la creación tendrás un fichero **package.json** como el siguiente:

```
{  
  "name": "ut5-npm-1",  
  "version": "1.0.0",  
  "description": "Primer package",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "Jesús Redondo García",  
  "license": "ISC"  
}
```

**NOTA:** Si quieres todos los valores por defecto en el package.json, puedes utilizar el siguiente comando:

```
npm init -y
```

# UT5.3.6 - NPM -Instalación de packages y módulos:

La principal ventaja de utilizar packages npm como proyectos es que nos permite instalar packages o módulos que vamos a utilizar en nuestro propio package.

**Hay varias maneras de instalar packages:**

Teniendo en cuenta el ámbito:

- **Local:** Los packages/módulos (a partir de ahora los llamaremos dependencias) sólo se instalan para este package en concreto y no son utilizables fuera de él. **Así van a ser la mayoría de instalaciones.** Es el modo por defecto. Pueden ser de dos tipos:
  - **Sólo para dev** (es decir, sólo para desarrollo): Son módulos que sólo tiene sentido empaquetar y descargar cuando estamos desarrollando la aplicación, pero en la versión final no se deben empaquetar. —> `npm install --save-dev mocha` ó `npm i -D mocha`
  - **Dependencias necesarias para la ejecución:** Son módulos que necesitas para ejecutar la aplicación. —> `$ npm install --save bootstrap` ó `$ npm i -S bootstrap`
- **Global:** Pueden ejecutarse desde cualquier sitio de nuestro sistema, podemos entenderlos como extensiones de nuestra línea de comandos. **Mejor evitarlos en la medida de lo posible:**

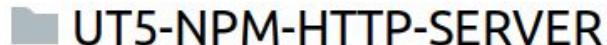
```
sudo npm install --global npm
```

```
sudo npm i -g npm
```

# UT5.3.6.1 - NPM -Instalación Global:

**Ejemplo-1** : Vamos a instalar un servidor http de manera local para developer en un package npm propio. Hay distintos servidores, por sencillez para este ejemplo, he elegido [http-server](#).

- 1) Crear una carpeta nueva donde se alojará el proyecto:



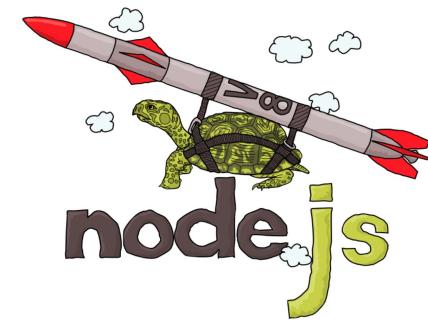
- 1) Empleando la consola de PHPStorm, ejecutar como superusuario:

```
'UT5-NPM-HTTP-SERVER$ npm init -y'
```

Para iniciar la carpeta como un package npm mediante el package.json

- 1) Instalamos la dependencia de http-server:

```
'UT5-NPM-HTTP-SERVER$ npm i -D http-server'
```



# UT5.3.6.1 - NPM -Instalación Global:

- 4) Creamos un script dentro de package.json para lanzar el servidor:

```
{  
  "name": "UT5-NPM-HTTP-SERVER",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "lanzarserver": "http-server"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "http-server": "^0.11.1"  
  }  
}
```

- 4) Lanzamos el servidor mediante el comando creado:

**UT5-NPM-HTTP-SERVER\$** npm run lanzarserver

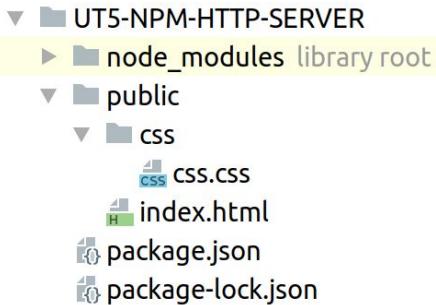
**Index of /**

📄 (drwxrwxr-x)	<a href="#">node_modules/</a>
📄 (-rw-rw-r--)	6.9k <a href="#">package-lock.json</a>
📄 (-rw-rw-r--)	325B <a href="#">package.json</a>

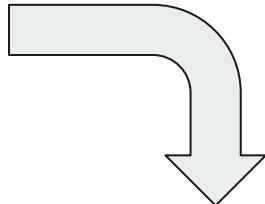
Node.js v11.0.0/ [ecstatic](#) server running @ 127.0.0.1:8080

# UT5.3.6.1 - NPM -Instalación Global:

- 6) Crea el index.html en **./public** y comprueba cómo funciona correctamente:



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Probando http-server</title>
</head>
<body>
  <h1>Página servida por http-server</h1>
</body>
</html>
```



**IMPORTANTE:** Por defecto nuestros navegadores cachean las páginas web durante el tiempo que indique el servidor.

Por defecto en http-server:

-c Set cache time (in seconds) for cache-control max-age header, e.g. -c10 for 10 seconds (defaults to '3600'). To disable caching, use -c-1.

**Soluciones:**

- 1) Llamar al servidor con “-c-1”
- 2) Abrir la ventana de depuración de Chrome. Mientras esté abierta, siempre se recarga la página sin usar caché  
CTRL-F5 (refresco sin caché)

## UT5.3.6.2 - NPM -Instalación Local:

**Ejemplo-2:** Volvemos a nuestra terminal, en la que creamos nuestro package... vamos a probar a instalar en local el módulo eslint.

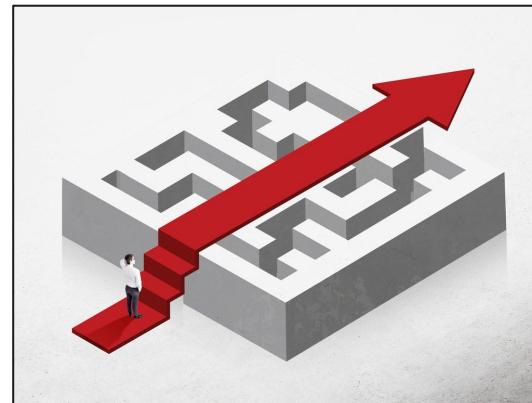
Este módulo se utiliza para checkear que los ficheros js siguen ciertas normas de estilo y nos devuelve información si no se cumplen. Obviamente, este módulo se va a instalar en local y además sólo para desarrollo (no es una dependencia que necesite nuestro package para funcionar, es algo que nos ayuda en el desarrollo). El comando es el siguiente:

```
npm install eslint --save-dev
```

ó abreviando:

```
npm i -D eslint
```

Click en la imagen para aprender sobre shorcuts de NPM:



## UT5.3.6.2 - NPM -Instalación Local:

**Ejemplo-Local:** Si nos fijamos ahora en el directorio en el que hemos hecho la instalación:

```
{  
  "name": "UT5-NPM-HTTP-SERVER",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "lanzarserver": "http-server"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "eslint": "^6.6.0",  
    "http-server": "^0.11.1"  
  }  
}
```

Tenemos dependencias en nuestro proyecto que son exclusivas para el desarrollo.

Estas dependencias se encuentran dentro de la propiedad *devDependencies*. En este caso vamos a utilizar "eslint" en la versión 5 y subversión 6.0 ó superior.

**NOTA:** Cuando hicimos un:  
*npm install*  
al iniciar el servidor lo que realmente ocurre es que se ven las dependencias del proyecto y las descarga.

**PRUEBA:** Elimina la carpeta:  
*/node\_modules*  
y vuelve a ejecutar:  
*npm i*

# UT5.3.6.2 - NPM -Instalación Local:

Recopilando... Desde npm podemos ver las dependencias instaladas...

Tanto en este package (en local):

```
jesus@jesus-x470:~/Dropbox/19-20-IESValleJerte/DAW - DWEC/UT5/19-20/UT5-NPM-HTTP-SERVER$ npm list --depth 0
UT5-NPM-HTTP-SERVER@1.0.0 /home/jesus/Dropbox/19-20-IESValleJerte/DAW - DWEC/UT5/19-20/UT5-NPM-HTTP-SERVER
└── eslint@6.6.0
└── http-server@0.11.1
```

Como en el equipo (en global):

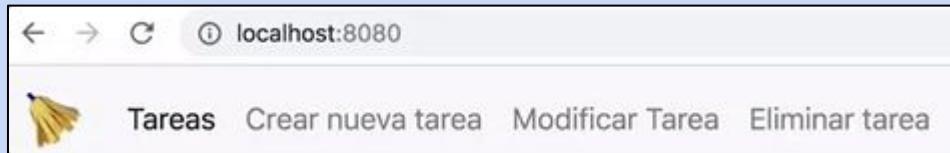
```
jesus@jesus-x470:~/Dropbox/19-20-IESValleJerte/DAW - DWEC/UT5/19-20/UT5-NPM-HTTP-SERVER$ npm list --global --depth 0
/usr/local/lib
└── @angular/cli@7.1.4
└── eslint@5.8.0
└── http-server@0.11.1
└── n@2.1.12
└── npm@6.13.0
└── uglify-js@3.4.9
└── uglifyjs@2.4.11
```

**NOTA:** Utilizamos --depth 0 para evitar que aparezcan dependencias de dependencias.

# UT5.3.6 - NPM - EJERCICIO 7:

**Ejercicio 7:** Este ejercicio consistirá en copiar el ejercicio anterior completo (el 6) al nuevo directorio con el que estamos trabajando dentro de la subcarpeta **/public**. Os recomiendo que para practicar copiéis todos los archivos necesarios mediante comandos de consola.

- Ahora prueba a lanzar utilizando el script que creamos anteriormente “**npm run lanzarserver**” desde consola.
- Si hubiese errores con la navegación de las páginas, arréglalos.
- Debe seguir funcionando correctamente el header, marcando la página actual:



---

**LAS SIGUIENTES  
DIAPOSITIVAS NO SE  
INCLUYEN EN EL  
TEMA**

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Como hemos mencionado anteriormente, hemos instalado **eslint** como dependencia de desarrollador para este **package**.

En este ejercicio debes leer la documentación de eslint y configurarlo para pasar un test de código sobre el fichero Tarea.js de javascript que hemos creado anteriormente. (Pulsa en la imagen para ir a la documentación).

### ESLint

[Website](#) | [Configuring](#) | [Rules](#) | [Contributing](#) | [Reporting Bugs](#) | [Code of Conduct](#) | [Twitter](#) |  
[Mailing List](#) | [Chat Room](#)

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. In many ways, it is similar to JSLint and JSHint with a few exceptions:

- ESLint uses [Espree](#) for JavaScript parsing.
- ESLint uses an AST to evaluate patterns in code.
- ESLint is completely pluggable, every single rule is a plugin and you can add more at runtime.

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Pasos:

- 1) Instalar **eslint** como developer para el package con el que estamos trabajando (ya debería estar hecho).
- 2) Crear un **script** para **lanzar la configuración de eslint**:

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  "lanzarserver": "http-server",  
  "eslint-config": "eslint --init"  
},
```

- 3) Lanzar el script y empezar la configuración:

? How would you like to use ESLint?

To check syntax only

To check syntax and find problems

› To check syntax, find problems, and enforce code style

Queremos que comprobar la sintaxis, encontrar problemas y además que sigamos un estilo determinado de codificación.

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Pasos:

- 4) Seleccionamos el tipo de módulos del proyecto. Por ahora irrelevante.

? What type of modules does your project use?  
JavaScript modules (import/export)  
> CommonJS (require(exports))  
None of these

- 4) No empleamos **ningún framework**:

? Does your project use TypeScript? > No / Yes

? Which framework does your project use?  
React  
Vue.js  
> None of these

- 4) Nuestro código corre en el navegador:

? Where does your code run? (Press **<space>** to select, **<a>** to toggle all, **<i>** to invert selection)  
 Browser  
 Node

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Pasos:

- 7) Vamos a crear directamente nosotros **nuestro estilo de codificación**:

? How would you like to define a style for your project?  
Use a popular style guide  
➤ Answer questions about your style  
Inspect your JavaScript file(s)

- 7) Lo vamos a configurar en **javascript**:

? What format do you want your config file to be in? **JavaScript**

- 7) Empleamos **tabulaciones** para “tabular” el código.

? What style of indentation do you use?  
➤ Tabs  
Spaces

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Pasos:

- 10) : Emplearemos comillas simples para las cadenas:

```
? What quotes do you use for strings?  
Double  
> Single
```

- 10) : En mi caso el fin de línea es Unix ¡¡¡Vosotros probad con WINDOWS!!!!

```
? What line endings do you use? (Use arrow keys)  
> Unix  
Windows
```

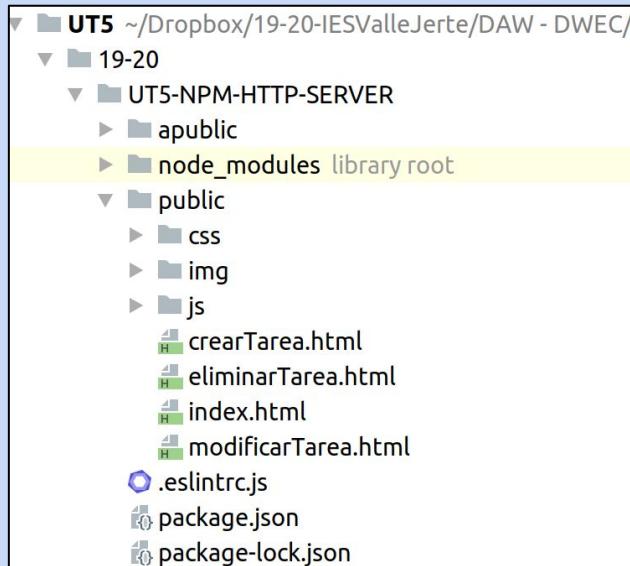
- 10) : Por último, indicamos que no necesitamos punto y coma al final de las sentencias.

```
? Do you require semicolons? (Y/n) n
```

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

Finalmente tenemos un fichero de configuración de eslint del tipo:



```
module.exports = {  
  'env': {  
    'browser': true,  
    'commonjs': true,  
    'es6': true  
  },  
  'extends': 'eslint:recommended',  
  'globals': {  
    'Atomics': 'readonly',  
    'SharedArrayBuffer': 'readonly'  
  },  
  'parserOptions': {  
    'ecmaVersion': 2018  
  },  
  'rules': {  
    'indent': [  
      'error'  
    ]  
  }  
};
```

The image shows a file explorer window with the following directory structure:

- UT5 (~/Dropbox/19-20-IESValleJerte/DAW - DWEC/UT)
- 19-20
  - UT5-NPM-HTTP-SERVER
    - apublic
    - node\_modules library root
    - public
      - css
      - img
      - js
        - crearTarea.html
        - eliminarTarea.html
        - index.html
        - modificarTarea.html
  - .eslintrc.js
  - package.json
  - package-lock.json

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

LLevar a cabo las pruebas de estilo:

- 1) Creo el script eslint:

```
"eslint": "eslint"
```

**IMPORTANTE:** Se le pueden pasar argumentos al script mediante el uso de los dos guiones: --

- 1) Lo invoco con el nombre del fichero que deseo comprobar. EJ:



```
UT5-NPM-HTTP-SERVER$ npm run eslint -- ./public/js/Tarea.js
```

- 1) Obviamente, tengo erres (muchísimos, porque las tabulaciones por defecto usan espacios, por ejemplo):

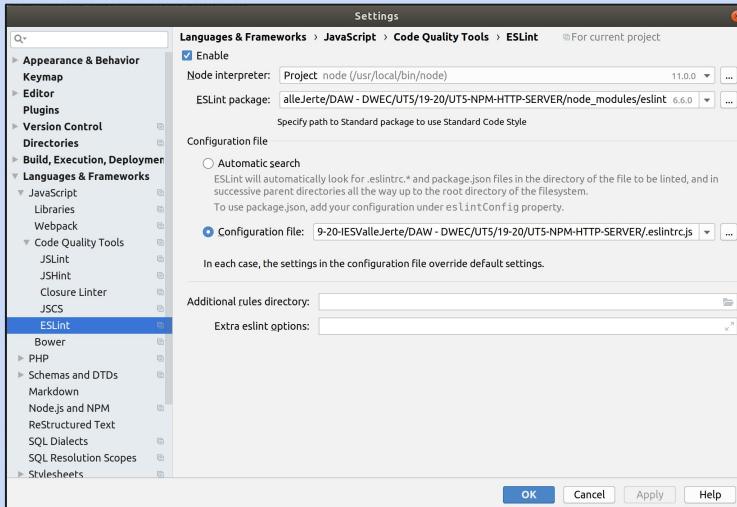
```
1:13  error  Extra semicolon                      semi
6:7   error  'Tarea' is defined but never used    no-unused-vars
8:1   error  Expected indentation of 1 tab but found 4 spaces  indent
9:1   error  Expected indentation of 2 tabs but found 8 spaces  indent
9:15  error  Strings must use singlequote         quotes
9:45  error  Extra semicolon                      semi
```

# UT5.3.6 - NPM - EJERCICIO 8:

## Ejercicio 8: Probar eslint.

LLevar a cabo las pruebas de estilo:

- 4) Podemos ejecutar eslint desde PHPStorm (ojo, en la versión 2018 no funciona). Pulsa Ctr-Alt-S y configura:



# UT5.3.6 - NPM - EJERCICIO 8:

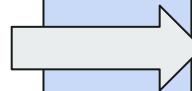
## Ejercicio 8: Probar eslint.

LLevar a cabo las pruebas de estilo:

- 5) Por último, podemos arreglar los estilos haciendo uso de eslint. Por ejemplo para arreglar lo que se pueda de mi fichero “Tarea.js”:

**UT5-NPM-HTTP-SERVER\$ npm run eslint -- ./public/js/Tarea.js --fix**

```
class Tarea{  
    static urlTarea(){  
        return "http://localhost:3000/tasks/";  
    }  
  
    constructor(tareaDeServer){  
        this.id = tareaDeServer._id;  
        this.titulo = tareaDeServer.titulo;  
        this.descripcion = tareaDeServer.descripcion;  
        this.fecha_creacion = new Date(tareaDeServer.fecha_creacion);  
        this.estado = tareaDeServer.estado;  
  
        //Otra manera:  
        /*for (let nombreAtt in tareaDeServer){  
            this[nombreAtt] = tareaDeServer[nombreAtt];  
        }*/  
    }  
}
```



```
class Tarea{  
    static urlTarea(){  
        return 'http://localhost:3000/tasks/'  
    }  
  
    constructor(tareaDeServer){  
        this.id = tareaDeServer._id;  
        this.titulo = tareaDeServer.titulo;  
        this.descripcion = tareaDeServer.descripcion;  
        this.fecha_creacion = new Date(tareaDeServer.fecha_creacion);  
        this.estado = tareaDeServer.estado;  
  
        //Otra manera:  
        /*for (let nombreAtt in tareaDeServer){  
            this[nombreAtt] = tareaDeServer[nombreAtt];  
        }*/  
    }  
}
```

## UT5.3.7 - NPM - Scripts:

Desde nuestro proyecto/package NPM podemos configurar una serie de Scripts para que los podamos lanzar con `npm run <nombredelscript>`. Esto es muy potente porque nos habilita crear proyectos que además tengan una serie de tareas que puedan ser lanzadas.

En el ejercicio 7 vimos cómo añadir un script editando **package.json**. Para ello basta con añadir un atributo en el objeto “scripts”:

```
"scripts": {  
  "test": "echo \\"$Error: no test specified\\" && exit 1",  
  "lanzar": "http-server"  
},
```

Podemos crear más scripts añadiendo más líneas. Pero también se pueden crear scripts que se ejecuten **antes o después** de otros scripts que ya hayamos creado. Para ello tenemos que nombrarlos igual pero con el prefijo **pre y post**. Por ejemplo, si queremos que antes de “lanzar” se abra el navegador para que veamos la web que estamos creando hay que crear un script “prelanzar”:

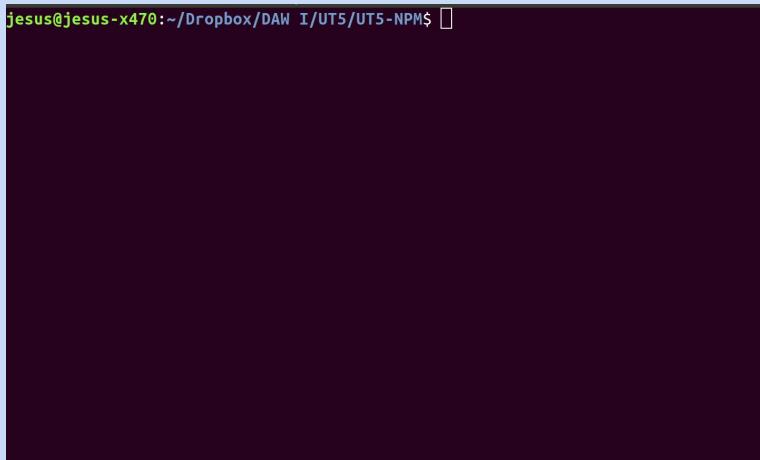
```
"scripts": {  
  "test": "echo \\"$Error: no test specified\\" && exit 1",  
  "lanzar": "http-server",  
  "prelanzar": "google-chrome http://localhost:8080/"  
},
```

**NOTA:** Si usas Windows... tendrás que buscar como se lanza google desde consola.

## UT5.3.7 - EJERCICIO 9:

---

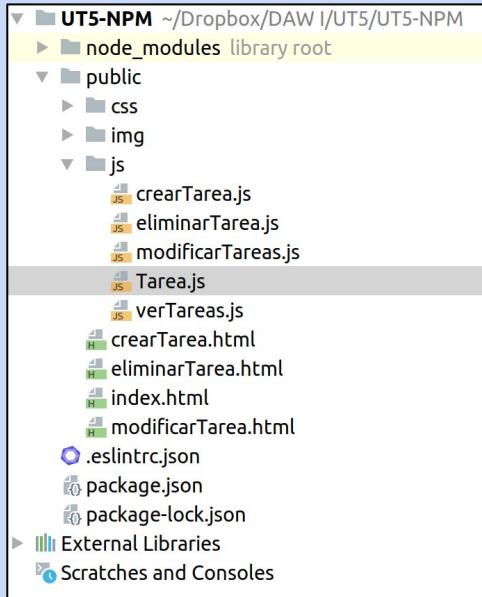
**Ejercicio 9:** Modifica el script “*test*” que viene por defecto para que pase **eslint** en todos los ficheros js de nuestro proyecto que estén dentro de la carpeta /public/js :



**NOTA:** Es posible (muy posible) que falle, ya que encuentra errores en el estilo de nuestros ficheros.

# UT5.3.7 - EJERCICIO 10:

**Ejercicio 10:** Abre el proyecto que hemos estado editando desde PHPStorm:



- **Eslint:** Por defecto lo detecta PHPStorm con la configuración que habíamos aplicado. Mejor desactivarlo, al menos por ahora que tenemos errores de variables no utilizadas.
- Podemos ejecutar los comandos con el terminal integrado de PHPStorm.
- **Crea un script denominado “limpiar-estilos” que nos “estilice” el código utilizando eslint con la configuración que hemos definido. Ejecútalo para comprobar que funciona.**

# UT5.4 - WEBPACK:

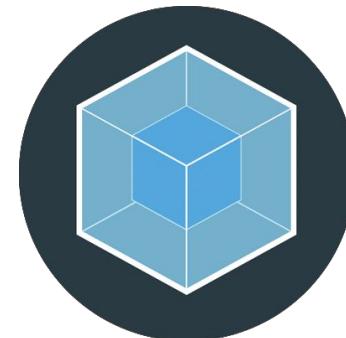
¿Qué es Webpack? Es un creador de lotes de módulos estáticos. Esto significa que en su funcionalidad principal, lo que hace webpack es coger partes estáticas de nuestra web (css,js,pngs...) y los empaquetarlas para su distribución.

Cuando Webpack procesa tu aplicación, internamente construye un grafo de dependencia que marca cada módulo que tu proyecto necesita y genera uno o más lotes/bundles.

Desde la versión 4.0.0 Webpack no necesita obligatoriamente un fichero de configuración para empaquetar el proyecto. Aun así, es muy interesante aprender a configurarlo para poder definir nuestras necesidades.  
Este fichero siempre se encuentra en **./webpack.config.js**.

Para comenzar con Webpack vamos a realizar una revisión teórica sobre sus **conceptos fundamentales**:

- |                        |                                   |
|------------------------|-----------------------------------|
| + Entrada(s) (entry)   | + Salida (output)                 |
| + Cargadores (loaders) | + Plugins                         |
| + Modo (mode)          | + Compatibilidad con el navegador |



## UT5.4.1 - Entrada (Entry)

---

**ENTRY:** Un punto de entrada (entry) indica qué módulo debe utilizar webpack para empezar a construir su grafo interno de dependencias. Webpack será capaz de resolver qué otros módulos y librerías ese punto de entrada necesita directa o indirectamente.

Por defecto, se configura un único punto de entrada que se encuentra en la ruta del proyecto:

`./src/index.js`

Pero podemos configurar uno distinto o incluso especificar varios puntos de entrada configurando la propiedad **entry** del fichero `./webpack.config.js`. Por ejemplo:

```
module.exports = {  
  entry: './path/to/my/entry/file.js'  
};
```

## UT5.4.2 - Salida (Output)

---

**OUTPUT:** La propiedad “output” le comunica a webpack dónde deben emitirse el/los bundle/s que va a crear y cómo debe llamar a esos ficheros.

Por defecto, el punto de salida para el punto de entrada principal es:

`./dist/main.js`      y la carpeta      `./dist`      para cualquier otro fichero generado.

Puedes configurar la salida en el fichero **webpack.config.js**:

```
const path = require('path');

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js'
  }
};
```

En este ejemplo estamos utilizando **output.filename** y **output.path** para decir a webpack el nombre de nuestro bundle y dónde lo queremos emitir.

En caso de que te estés preguntando por qué estamos importando **path** en la primera línea. Es un módulo de node.js que nos permite trabajar y manipular rutas (paths) de manera independiente al S.O. que utilicemos.

## UT5.4.3 - Cargadores (Loaders)

---

**LOADERS:** Sin ninguna configuración, Webpack sólo entiende Javascript y ficheros JSON. Los Loaders permiten a Webpack procesar otros tipos de ficheros y convertirlos en módulos que pueden ser consumidos por tu aplicación y añadidos al [grafo de dependencias](#).

Esta es una característica única de Webpack, la de importar cualquier tipo de módulos (por ejemplo ficheros css. Esta extensión del lenguaje está justificada porque permite a los desarrolladores crear un grafo de dependencias más preciso.

A un alto nivel, los **loaders** tienen dos propiedades en el fichero **webpack.config.js**:

- 1) **test**: Es una expresión regular que identifica qué ficheros deben ser procesados.
- 1) **use**: Indica qué cargador se debe utilizar.

## UT5.4.3 - Cargadores (Loaders)

---

### LOADERS:

```
const path = require('path');

module.exports = {
  output: {
    filename: 'my-first-webpack.bundle.js'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  }
};
```

La configuración de la izquierda tiene definida la propiedad **rules** para un sólo módulo que tiene dos propiedades: “**test**” y “**use**”.

Esta configuración está diciendo a webpack lo siguiente:

*“Oye, compilador de Webpack”, cuando te cruces con una ruta que se resuelve como un “.txt” dentro de una sentencia de tipo `require/import`, utiliza el loader “raw-loader para transformarlo antes de añadirlo al bundle.*

**NOTA:** [Más info sobre loaders aquí.](#)

## UT5.4.4 - Plugins

**PLUGINS:** Mientras que los cargadores son útiles para transformar ciertos tipos de módulos, los plugins se pueden emplear para llevar a cabo una serie de tareas más diversas como puede ser: optimización de los bundles, manejo de assets o inyección de variables de entorno.

En el [apartado plugins de la documentación](#) podrás encontrar información relevante sobre Plugins.

Para usar un plugin debes hacer un **require** y añadirlo al array de plugins. La mayoría de plugins son personalizables a través de sus opciones. Dado que puedes utilizar un plugin múltiples veces en el fichero **webpack.config.js** para distintos propósitos, necesitas crear una instancia del plugin con la palabra reservada **new**.

```
const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm
const webpack = require('webpack'); //to access built-in plugins

module.exports = {
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};
```

En este ejemplo, el plugin *html-webpack-plugin* genera un fichero HTML para tu aplicación, inyectando automáticamente todos los bundles generados.

Puedes ver una lista de plugins recomendados [AQUÍ](#).

## UT5.4.5 - Modo (Mode)

---

**MODE:** Definiendo el parámetro **mode** a alguno de los valores válidos, que son “development”, “production” ó “none”, podemos activar las optimizaciones correspondientes para cada entorno.

El valor por defecto es “producción”, “production”.

```
module.exports = {  
  mode: 'production'  
};
```

Puedes consultar más información sobre los modos [AQUÍ](#).

# UT5.4.6 - Compatibilidad del navegador

Webpack soporta todos los navegadores que manejan ECMAScript-5 (la versión 2009 de Javascript).

Que en la actualidad son la gran mayoría de navegadores (de IE8 hacia abajo no están soportados).

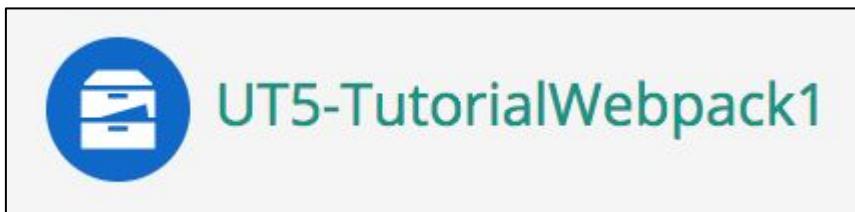
Feature name	Current browser	Desktop browsers																		Servers/runtimes										Mobile	
		Compilers/polyfills		66%	99%	100%	100%	100%	100%	100%	100%	100%	100%	97%	99%	99%	84%	96%	79%	97%	98%	99%	100%	0%	99%	98%	98%	100%	Mobile		
Object/array literal extensions	5/5	0/5	3/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5	5/5		
Object static methods	13/13	1/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	0/13	13/13	13/13	13/13	13/13	13/13		
Array methods	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	11/12	12/12	0/12	12/12	12/12	12/12	12/12	12/12	
String properties and methods	2/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2
Date methods	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	3/3	3/3	3/3	3/3	3/3	
Function.prototype.bind	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes		
JSON	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes																			
Immutable globals	3/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	3/3	3/3	3/3	3/3	3/3		
Miscellaneous	8/8	0/8	7/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	7/8	7/8	7/8	7/8	7/8	7/8	7/8	7/8	7/8	7/8	0/8	7/8	8/8	8/8	8/8	7/8		
Strict mode	19/19	0/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	0/19	19/19	19/19	17/19	17/19	19/19		

La gran mayoría de navegadores están soportados. Por supuesto, todos los navegadores modernos están incluidos.

# UT5.4.7 - Webpack - iPruebas!

---

Descarga del campus virtual el proyecto base con el que vamos a trabajar con Webpack:



## ¿Qué vamos a hacer?

- Inicializar el proyecto como package NPM.
- Cargar la dependencia webpack para desarrollo.
- Crear un script denominado “build” que ejecute Webpack. En esta primera versión le pasaremos el punto de entrada directamente desde el script:
- Prueba el proceso de build.
- Gestionar el proceso de build mediante el fichero de configuración webpack.config.js

# UT5.4.7 - Webpack - iPruebas!

---

- 1) Inicializar el proyecto como package NPM.

Terminal

```
+ jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm init -y
```

- 1) Cargar la dependencia webpack para desarrollo.

Terminal

```
+ jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm install webpack webpack-cli --save-dev
```

- 3) Crear un script denominado “build” que ejecute Webpack. En esta primera versión le pasaremos el punto de entrada directamente desde el script:

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "build": "webpack ./src/js/app.js"  
},
```

# UT5.4.7 - Webpack - iPruebas!

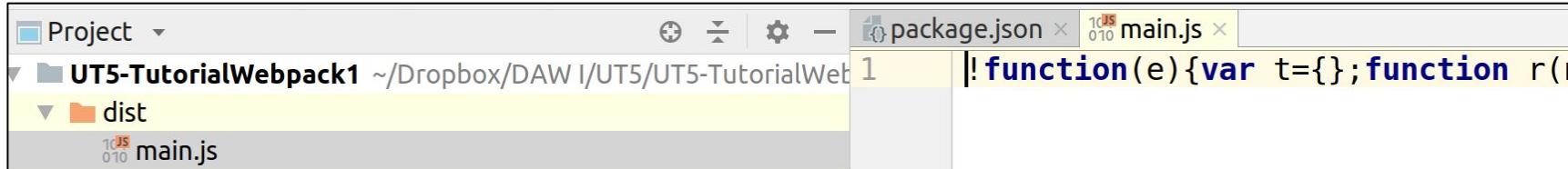
MUY IMPORTANTE: Todas las configuraciones que podemos emplear en la línea de comandos:  
<https://webpack.js.org/api/cli/>

## 4) Prueba el proceso de build:

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm run build
```

Se ha creado un bundle con toda la información de la web. Teniendo en cuenta que hemos indicado la entrada en la llamada del script con “src/js/app.js”, el bundle contiene ahora mismo sólamente ese fichero.

El bundle por defecto se crea en el directorio /dist/main.js:



The screenshot shows a code editor interface. On the left is a sidebar labeled "Project" with a dropdown arrow. Below it, under the project name "UT5-TutorialWebpack1", there is a folder icon labeled "dist" and a file icon labeled "main.js". The main workspace has two tabs: "package.json" and "main.js". The "package.json" tab shows a JSON object with a "scripts" key containing three entries: "test", "build", and "build:dev". The "main.js" tab shows the contents of the generated JavaScript file, which is minified and starts with the line "1 !function(e){var t={};function r(r".

Por defecto webpack corre en modo “production” por lo que la versión del bundle está minificada. Puedes crear otro script para generar el build como “development” y comprobar como no se minifica:

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "build": "webpack ./src/js/app.js",  
  "build:dev": "webpack ./src/js/app.js --mode=\"development\""  
},
```



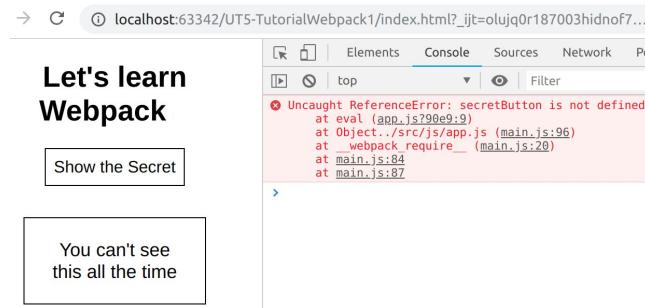
# UT5.4.7 - Webpack - iPruebas!

## 4) Prueba el proceso de build:

Falta indicar que utilizamos el script “bundle.js” en lugar de los scripts “dom-loader.js” y “app.js”. Modificamos el index.html:

```
<p id="secret-paragraph>You can't see this all the time</p>
<script src="dist/main.js"></script>
<!--<script src="src/js/dom-loader.js"></script>
<script src="src/js/app.js"></script>-->
</body>
```

Si probamos la web... ¡No funciona!



**NOTA:** Es normal que no funcione. Para crear el bundle.js sólo estamos empleando el fichero “**src/js/app.js**”. ¡No podemos acceder a las variables declaradas en “**src/js/dom-loader.js**”.

# UT5.4.7 - Webpack - iPruebas!

---

## 4) Prueba el proceso de build:

Para que podamos utilizar las variables desde dom-loader.js a app.js tenemos que exportarlas y requerirlas/importarlas.

**En el fichero dom-loader.js las podemos exportar de las siguientes maneras (son equivalentes):**

### 1) Mediante exports:

```
exports.secretButton = document.querySelector('#secret-button');
exports.secretParagraph = document.querySelector('#secret-paragraph');
```

### 2) Mediante module.exports:

```
module.exports={
  secretButton : document.querySelector('#secret-button'),
  secretParagraph:document.querySelector('#secret-paragraph')
};
```

### 3) Mediante export:

```
export let secretButton = document.querySelector('#secret-button');
export let secretParagraph = document.querySelector('#secret-paragraph');
```

# UT5.4.7 - Webpack - iPruebas!

---

## 4) Prueba el proceso de build:

Ahora debemos importarlas dentro de app.js. Existen también dos maneras de importarlas:

### 1) Con require:

```
let obj = require("./dom-loader.js");  
  
let secretButton = obj.secretButton;  
let secretParagraph = obj.secretParagraph;
```

### 2) Con import:

```
import {secretButton, secretParagraph} from "./dom-loader";
```

**NOTA:** Vuelve a hacer build y comprueba que efectivamente funciona:

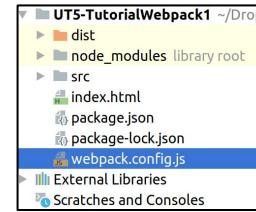


**PREGUNTA:** Hemos dicho que desde el navegador no se puede realizar la sentencia “require” ni import...  
¿Cómo es posible que ahora la página web funcione?

# UT5.4.7 - Webpack - iPruebas!

## 5) Gestionar el proceso de build mediante el fichero de configuración webpack.config.js:

Creamos un fichero llamado webpack.config.js en el directorio del proyecto:



Y modificamos su contenido para indicar cuál es el punto de entrada y de salida de webpack:

```
let path = require("path");
module.exports = {
  entry: "./src/js/app.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "main.js",
    //publicPath: 'dist/' //<<<---- OJO, Con HtmlWebpackPlugin no puedo emplearlo.
  }
};
```

Por último, actualizamos nuestro script.

Ya no se necesita saber dónde está el punto de entrada:

```
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1",
  "build": "webpack",
  "build:dev": "webpack --mode=\"development\""
},
```

# UT5.4.7 - Webpack - iPruebas!

---

## 6) Automatizar la inclusión del js mediante el plugin html-webpack-plugin:

Instalamos el plugin: `/UT5-TutorialWebpack1$ npm install --save-dev html-webpack-plugin`

Dicho plugin permite cargar los html como propios templates e incluir los fragmentos de js que necesitemos automáticamente. Como en este ejemplo sólo tenemos un html y un punto de entrada. La configuración es extremadamente sencilla:

```
let path = require('path');
let HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: "./src/js/app.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename:"main.js"
    //publicPath: dist/ //<-- OJO No emplear con HtmlWebpackPlugin
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html',
      filename: 'indexGenerado.html'
    })
  ]
};
```

**Nota:** El html se genera en ./dist aunque desafortunadamente no funcionan los estilos:

**Let's learn Webpack**

Show the Secret

# UT5.4.7 - Webpack - iPruebas!

## 7) Incluir la carga del css mediante css-loader :

Falta especificar los loaders en el fichero **webpack.config.js**. El modo de especificarlo es el siguiente:

```
let path = require('path');
let HtmlWebpackPlugin = require ('html-webpack-plugin');
module.exports = {
  entry : './src/js/app.js',
  output : {
    path : path.resolve(__dirname, 'dist'),
    filename: 'salida.js',
    //publicPath: 'dist/' //<---- OJO, no necesario con webpack nuevo
  },
  module: {
    rules:[
      {
        test: /\.css$/,
        use:[
          'style-loader',
          'css-loader'
        ]
      }
      //<-- Más reglas de loaders aquí
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html',
      filename: 'MiIndex.html'
    })
  ]
};
```

Lo nuevo que estamos añadiendo viene a ser lo siguiente:

*"Webpack, vamos a utilizar unas reglas para definir cómo tratar ciertos archivos. Cada una de esas reglas es un objeto dentro de **module.rules**.*

*En este caso, para todos los ficheros que terminen en ".css" les aplicaremos primero el css-loader y después el style-loader (¡¡¡ojo que el orden al describirlo es inverso!!)."*

# UT5.4.7 - Webpack - iPruebas!

---

## 7) Incluir la carga del css mediante css-loader :

Ahora vamos a cargar otros archivos estáticos que no son ficheros js, vamos a importar ficheros css. Para ello debemos emplear un **loader**. Por defecto, Webpack no sabe cómo convertir un fichero CSS y guardararlo en el bundle, esta acción se lleva a cabo en dos partes en el caso de los css:

- 1) Cargar el .css, procesarlo para que lo pueda cargar webpack. → **css-loader**
- 1) Emplear ese .css en el bundle. Qué pasos hay que dar para empaquetarlo. → **style-loader**

Para cargar esos loaders hay que instalarlos:

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm install css-loader style-loader --save-dev
```

El primer paso es cargar los css desde app.js:

```
import "../css/main.css";
import "../css/input-elements.css";
```

```
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<!-<link rel="stylesheet" href="src/css/main.css">->
<!-<link rel="stylesheet" href="src/css/input-elements.css">->
+ + + + + Webpack 2 Basic + + + + +
```

Y eliminarlos de index.html:

# UT5.4.7 - Webpack - iPruebas!

---

## 7) Incluir la carga del css mediante css-loader :

Ya se pueden importar en **app.js** los **css** necesarios como si de otros ficheros cualesquiera se tratara:

```
import './css/input-elements.css';
import './css/main.css';
import {secretButton, secretParagraph} from './dom-loader';
```

Ahora la web funciona con normalidad y con estilos:

Let's learn Webpack

Show the Secret

# UT5.4.8 - Webpack-dev-server

---

**Webpack-dev-server** es un paquete que funciona exactamente igual que webpack (es decir, es capaz de empaquetar recursos para su distribución) pero además incluye tres características extra:

- 1) Incluye un servidor para ejecutar las pruebas de código. Es decir, podemos tener un servidor http para las pruebas sin tener que utilizar el de por defecto de PhpStorm ni tampoco http-server.
- 1) El servidor provee ayuda para que se recargue automáticamente en bundle generado si hacemos modificaciones sobre los ficheros de entrada: **¡Muy importante para desarrollo!**
- 1) No es necesario crear los bundles en disco, se pueden cargar directamente desde memoria RAM.



# UT5.4.8 - Webpack-dev-server ¡Pruebas!

Qué vamos a hacer con Webpack-server:

## 1) Instalarlo:

Terminal

```
+ jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm install webpack-dev-server --save-dev
```

## 2) Modificar nuestros scripts, añadiendo uno para lanzar webpack dev server (**OJO HA CAMBIADO tras WEBPACK 5 y WEBPACK-CLI 4**).

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "launch": "webpack serve",  
  "build": "webpack",  
  "build:dev": "webpack --mode=\"development\""  
},
```

## 3) Comprueba que funciona, para ello, borra la carpeta /dist (recuerda que el servidor de webpack funciona en RAM) y **accede a la URL**:

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm run build
```

# UT5.4.8 - Webpack-dev-server iPruebas!

Qué vamos a hacer con Webpack-server:

## 4) El fichero webpack.config.js:

```
let path = require('path');
let HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: "./src/js/app.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename:"main.js"
    //publicPath: dist/ //<-- OJO No emplear con HtmlWebpackPlugin
  },
  module: {
    rules:[
      {
        test: /\.css$/, //Termina en CSS
        use:[
          'style-loader',
          'css-loader'
        ]
      }
    ],
    plugins: [
      new HtmlWebpackPlugin({
        template: './index.html',
        filename:'indexGenerado.html'
      })
    ]
  };
};
```

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

---

Ahora mismo nuestro index.html sólamente tiene un punto de entrada, que es app.js :

```
jesus@jesus-x470:~/Dropbox/DAW I/UT5/UT5-TutorialWebpack1$ npm run launch
```

Let's learn Webpack

Hide the Secret

You can't see this all  
the time



The screenshot shows a browser window displaying the source code of index.html. The code includes standard HTML tags like <html>, <head>, and <body>. It also contains several CSS imports and styles. A specific line of code is highlighted in grey: <title>Webpack 2 Basics</title> == \$0. This indicates that the loader has converted the original CSS imports into inline styles.

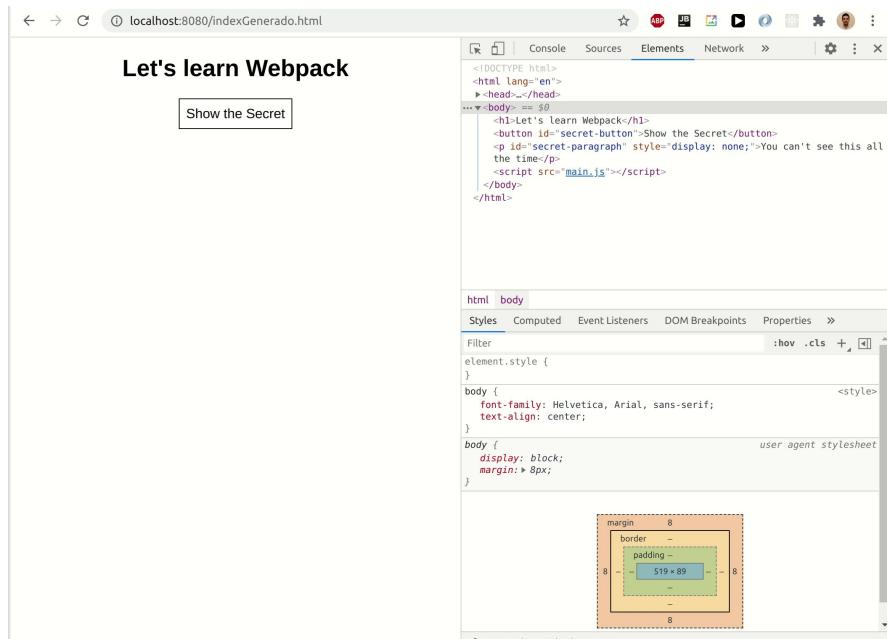
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <!--<link rel="stylesheet" href="src/css/main.css">-->
    <!--<link rel="stylesheet" href="src/css/input-elements.css">-->
    ...
    <title>Webpack 2 Basics</title> == $0
    ><style type="text/css"></style>
    ><style type="text/css"></style>
  </head>
  <body>...
    <div>You can't see this all the time</div>
  </body>
</html>
```

Toda nuestra web funciona con un solo script, que es main.js. Podemos ver cómo el loader ha cambiado los links de css por unos <styles>

Estas etiquetas preservan el orden original de los imports en el javascript para que se apliquen por cascada los estilos.

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

Puedes comprobar la importancia de utilizar Webpack y Webpack-dev-server con un ejemplo muy simple.  
Fíjate qué ocurre cuando modifico los estilos CSS :



Webpack-dev-server está escuchando por cambios en los ficheros del grafo de dependencias y automáticamente los recarga si hay cambios en los mismos.

# EJERCICIOS PARA ENTREGAR

---



# UT5.4.9 - Webpack-dev-server-Webpack - CSS

**Ejercicio FINAL:** Modifica tu proyecto de **TAREAS** para que todo el proyecto funcione mediante **Webpack** y **Webpack-dev-server**. Será necesario también tener un script de **eslint** para que todo el código del proyecto siga el mismo formato.

Para configurar webpack debéis indicar los puntos de entrada, de salida, los loaders y además es obligatorio emplear el plugin `html-webpack-plugin` para incrustar directamente los bundles a cada página web.

El fichero `package.json` debe tener los siguientes scripts, **EMPLEAD ESOS NOMBRES**:

- **servir-RAM**: Lanzará webpack-dev-server en el cual estará en ejecución en RAM (para asegurarlo elimina la carpeta “dist” y comprueba que funciona).
- **build-prod**: Lanzará webpack en modo producción creando todos los bundles necesarios para la distribución de la web a un servidor.
- **limpiar-estilos**: Lanzará **eslint** sobre todos los ficheros de la carpeta donde guardes los js con el parámetro **--fix**. El estilo que se aplicará será el siguiente:

*Va a forzar el uso de estilo // Se emplea CommonJS // No empleamos ningún framework // Código que corre en el navegador y en Node // Vamos a definir nuestro estilo mediante un fichero javascript // Indexamos mediante tabulaciones // Las cadenas con comillas simples // Sin puntos y coma.*

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

**Ejercicio FINAL:** Para poder configurar webpack en una web donde hay varias páginas es normalmente necesario establecer distintos puntos de entrada. En mi caso he establecido en el fichero de configuración de webpack los siguientes puntos de entrada:

Nombres de los puntos de entrada

```
entry: {  
    ver: "./public/js/verTareas.js",  
    crear: "./public/js/crearTarea.js",  
    eliminar: "./public/js/eliminarTarea.js",  
    modificar: "./public/js/modificarTareas.js",  
    cabecera: "./public/js/cabecera.js"  
},
```

Ubicación de los archivos de entrada

Y generar automáticamente varias salidas:

[name] es cada uno de los nombres dados

```
output : {  
    path : path.resolve(__dirname, 'dist'),  
    filename: '[name].js',  
},
```

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

**Ejercicio FINAL:** Ahora debes ejecutar tantas instancias de **HtmlWebpackPlugin** como páginas tengas (en mi caso 4):

```
plugins: [
  new HtmlWebpackPlugin({
    template: './public/verTareas.html',
    filename: 'index.html',
    chunks: ['ver']
  }),
  new HtmlWebpackPlugin({
    template: './public/crearTarea.html',
    filename: 'crearTarea.html',
    chunks: ['crear']
  }),
  new HtmlWebpackPlugin({
    template: './public/eliminarTarea.html',
    filename: 'eliminarTarea.html',
    chunks: ['eliminar']
  }),
  new HtmlWebpackPlugin({
    template: './public/modificarTarea.html',
    filename: 'modificarTarea.html',
    chunks: ['modificar']
  })
],
```

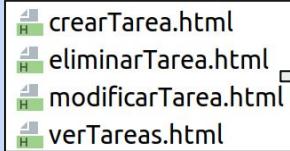
En el atributo “chunk” es cada una de los outputs que necesitas en cada una de las páginas.

[Tutorial sobre webpack con info sobre varias entradas/salidas](#)

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

## Ejercicio FINAL: Por último...

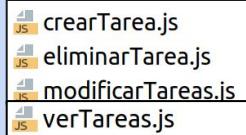
No olvides comentar/eliminar los scripts que tenías en los html:



```
<!--  
    <script src="js/Tarea.js"></script>  
    <script src="js/verTareas.js"></script>  
    <script src="js/cabecera.js"></script>  
-->
```

**NOTA:** Es posible que las imágenes no funcionen. No te preocupes, cuando lo distribuyas como prod, simplemente copia la carpeta img en dist.

Exporta e importa las dependencias que necesitan tus “entry points”.



```
import "../css/estilosTareas.css"  
import { rutaProyecto, cargarheader, esActive } from './cabecera'  
import { Tarea } from './Tarea'  
  
cargarheader()
```



**En mi caso debo hacer esto para cargar el header**

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

Ejercicio FINAL: Si quisiéramos cargar imágenes...

A partir de Webpack 5 es muy sencillo. Puedes leer la documentación en [Webpack - Loading Images](#). También puedes revisar [asset/resource](#), que es el loader que se está empleando.

```
{  
  test: /\.(png|svg|jpg|jpeg|gif)$/,  
  type: 'asset/resource',  
},
```

Incluye el loader correspondiente para cargar recursos y ficheros como URLs.

```
const Fregona = require('../img/fregona_tiras_resized.png');
```

Importa la imagen. Sólo funciona correctamente con **require**.

```
<a class="navbar-brand" href="#">  
    
</a>
```

Úsalo donde lo necesites para cargar la URL.

# UT5.4.9 - Webpack-dev-server-Webpack - CSS

## Ejercicio FINAL: Entrega

Entrega todo el proyecto en Moodle. No olvides eliminar la carpeta node\_modules:

Cualquier ejercicio entregado con node\_modules no será corregido y su nota final será de cero (0).

La entrega de la tarea se encuentra en Moodle:

