

DESENVOLVIMIENTO DE INTERFACES WEB 22_23_Ord

[Taboleiro](#) / [Os meus cursos](#) / [DIW 22 23 Ord](#) / [UD6. VUEJS](#) / [4. MEVN. MongoDB, Express, Vue y Node. Configuración del Frontend. \(II \)](#)

4. MEVN. MongoDB, Express, Vue y Node. Configuración del Frontend. (II)

Introducción

El paso siguiente es la configuración del **frontend**, en este caso con Vue.js.

Tenemos que tener en cuenta que, debido a que partimos de 0, tendremos que tocar ficheros de configuración.

En primer lugar instalamos **webpack**



```
Terminal: Local x + v
PS C:\Users\Carlos.PC-PEREZABALDE\WebstormProjects\mevn> npm install webpack -D

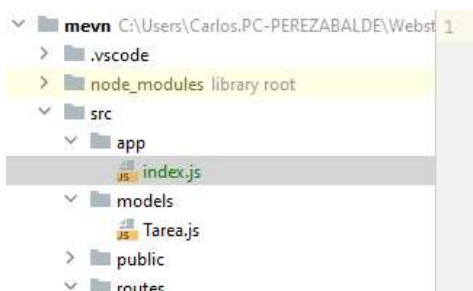
added 59 packages, and audited 319 packages in 19s

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

Webpack es un **empaquetador de módulos**, es decir, te **permite generar un archivo único** con todos aquellos módulos que necesita tu aplicación para funcionar. Para darte una idea, te **permite incluir todos tus archivos javascript .js** en un único archivo, **incluso se pueden incluir hasta archivos de estilos .css** en el mismo archivo, llamado *.bundle.js. Además se puede realizar otras tareas de optimización de los códigos, tales como la **minificación** y la **compresión**. La -D significa que es una dependencia en desarrollo (ver package.json *devdependencies*)

```
{
  "vue": "^3.2.45"
},
"devDependencies": {
  "@vitejs/plugin-vue": "^3.2.0",
  "vite": "^3.2.4",
  "webpack": "^5.75.0"
},
```

A continuación, y dentro de la carpeta **app**, vamos a crear un archivo, el **index.js**, puede tener otro nombre pero lo llamamos así porque será el archivo índice donde arranque la parte de la aplicación de frontend.



Antes de continuar hay que decir que el navegador no entiende de vue.js sino de **js**. Por lo tanto hay que configurar **package.json** para traducir ese código, y que lo haga en modo desarrollo para eso usamos **webpack**

```
"scripts": {
  "dev": "nodemon src/index.js",
  "build": "vite build",
  "preview": "vite preview",
  "webpack": "webpack --mode development"
},
```

Y en el **webpack.config.js** le decimos donde está el archivo a traducir y donde quieres que coloque la traducción o compilación y su nombre:

```
module.exports = {
  entry: './src/app/index.js',
  output: {
    path: __dirname + '/src/public',
    filename: 'bundle.js'
  }
};
```

Además necesito instalar un módulo llamado **webpack-cli** y también **-D** para indicar la dependencia en desarrollo:

```
WebstormProjects\mevn> npm i webpack-cli -D
index: timing reifyNode:node_modules/webpack-merge Completed in 1083ms
```

Y en **package.json**.

```
{
  "devDependencies": {
    "@vitejs/plugin-vue": "^3.2.0",
    "vite": "^3.2.4",
    "webpack": "^5.75.0",
    "webpack-cli": "^5.0.1"
  }
},
```

Vamos a instalar varios módulos de una atacada.

En primer los módulos como son **babel-loader**, **babel-preset-env** que se encargará también de traducir cualquier versión moderna de JS que cualquier navegador pueda entender.

Además, otros como **vue-loader** and **vue-template-compiler**. Todos ellos en versión **-D o development**.

```
mProjects\mevn> npm install babel-loader babel-preset-env vue-loader vue-template-compiler -D
```

Con esto tenemos lo que necesitamos de momento par iniciar el proyecto.

A continuación vamos a configurar el fichero **.babel.rc** y **webpack.config.js** tal como se muestra a continuación.

Empezamos con **webpack.config.js**

```

1  const { VueLoaderPlugin } = require('vue-loader');
2
3  module.exports={
4    entry: './src/app/index.js',
5    output: {
6      path: __dirname + '/src/public',
7      filename: 'bundle.js'
8    },
9    module: {
10     rules: [
11       {
12         test: /\.js$/,
13         exclude: /node_modules/,
14         use:{
15           loader: 'babel-loader'
16         }
17       },
18       {
19         test: /\.vue$/,
20         loader: 'vue-loader'
21       }
22     ]
23   },
24   plugins: [
25     new VueLoaderPlugin()
26   ]
27 }

```

Básicamente este fichero contiene todo lo necesario para traducir los ficheros JS moderno a ficheros JS que entienda cualquier navegador dando como resultado el fichero **bundle.js** en la carpeta **public**. Además permite traducir los ficheros **Vue** a ficheros que también conozca el navegador.

```

{
  // ...
}

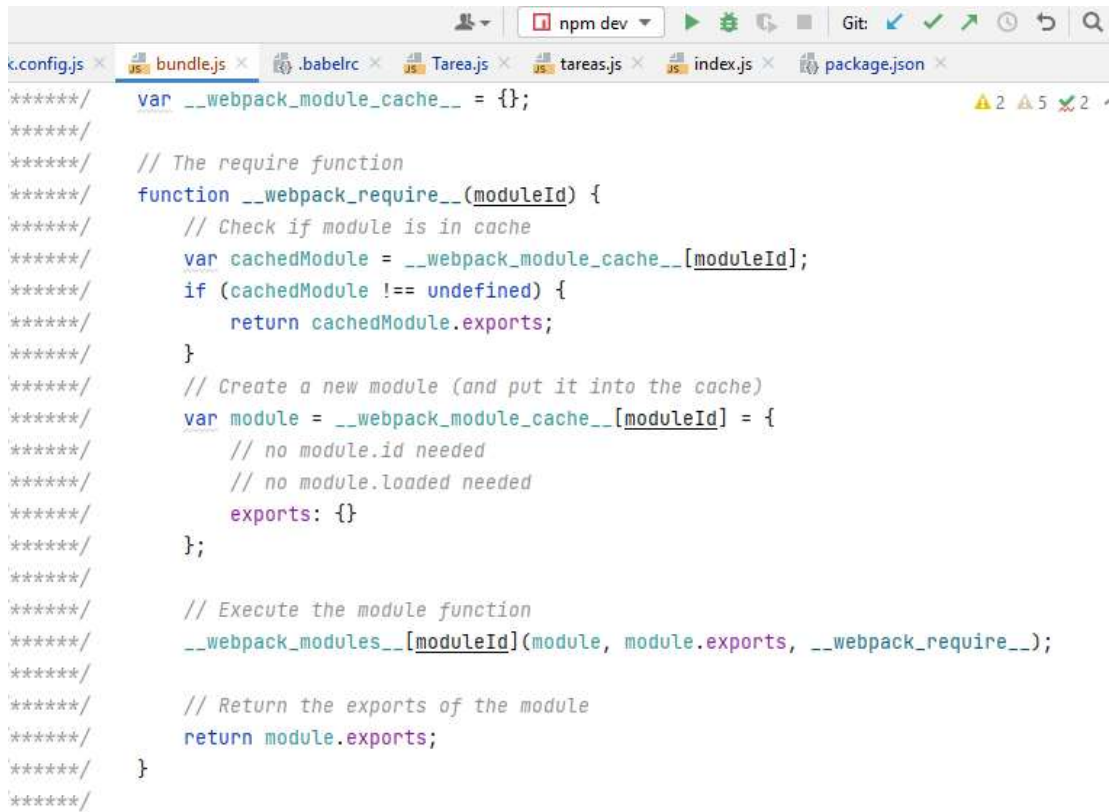
```

Hagamos una primera traducción para comprobar que todo está correcto.

```
PS C:\Users\Carlos.PC-PEREZABALDE\WebstormProjects\mevn> npm run webpack
```

```
> mevn@0.0.0 webpack
> webpack --mode development
```

```
asset bundle.js 2.27 KiB [emitted] (name: main)
./src/app/index.js 131 bytes [built] [code generated]
webpack 5.75.0 compiled successfully in 1657 ms
```



```

*****// var __webpack_module_cache__ = {};
*****//
*****// // The require function
*****// function __webpack_require__(moduleId) {
*****//     // Check if module is in cache
*****//     var cachedModule = __webpack_module_cache__[moduleId];
*****//     if (cachedModule !== undefined) {
*****//         return cachedModule.exports;
*****//     }
*****//     // Create a new module (and put it into the cache)
*****//     var module = __webpack_module_cache__[moduleId] = {
*****//         // no module.id needed
*****//         // no module.loaded needed
*****//         exports: {}
*****//     };
*****//
*****//     // Execute the module function
*****//     __webpack_modules__[moduleId](module, module.exports, __webpack_require__);
*****//
*****//     // Return the exports of the module
*****//     return module.exports;
*****// }
*****//

```

Si todo ha ido correcto nos aparece el fichero **bundle.js** que es el fichero JS traducido para que cualquier navegador pueda entenderlo.

Por otro lado preparamos el fichero **index.html** en la carpeta public para que lea el fichero o cargue **bundle.js**



```

<!DOCTYPE html>
<html lang="es">
  <head>
    <title>MEVN</title>
  </head>
  <body>
    <h1>BREVE MANUAL MEVN</h1>
    <script src="bundle.js"></script>
  </body>
</html>

```

Lanzamos el servidor con **npm run dev** y vemos que carga el fichero **bundle.js** que obviamente de momento no hace nada.

```

warning.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Servidor a la escucha en el puerto 3000
Base de datos conectada
GET / 200 34.743 ms - 172
GET /bundle.js 200 2.316 ms - 2326
GET /favicon.ico - - ms - -

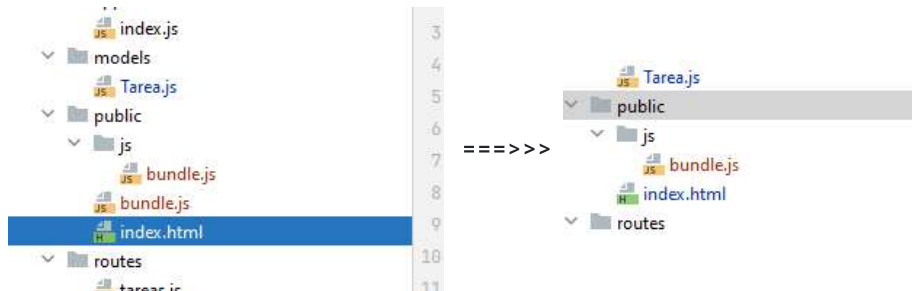
```

Hagamos algún cambio más. Vamos a colocar el fichero bundle.js en un subdirectorio dentro de public llamado js, para ir ordenando todo. Para ello modificamos el fichero **webpack.config.js**

```
entry: './src/app/index.js',
output: {
  path: __dirname + '/src/public/js',
  filename: 'bundle.js'
},
module: {
```

Volvemos a compilar con **webpack** y eliminamos el antiguo **bundle.js**, el resultado sería tras compilar.

```
ully in 1657 ms
E:\WebstormProjects\mevn> npm run webpack
```



No nos olvidemos de modificar **el index.html**.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>MEVN</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="/js/bundle.js"></script>
  </body>
</html>
```

Vamos ahora a automatizar algunas tareas de las que estamos haciendo para no tener que estar reiniciando. Para eso, en el **package.json** añadimos la opción **--watch** al script **webpack**. Así cada vez que hago un cambio en el frontend ya se recompila cada vez que se haga un cambio.

```
"preview": "vite preview",
"webpack": "webpack --mode development --watch"
```

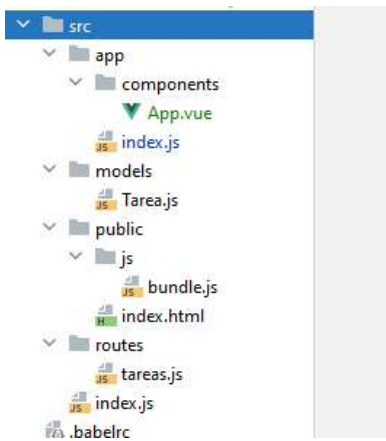
Muy bien. Ahora empecemos ya codificar el frontend. Para eso instalamos **vue**, si es necesario, ya que según el IDE que se utilice, puede o no ser necesario.

```
E:\WebstormProjects\mevn> npm install vue -D
dependencies.vue in favor of devDependencies.vue
```

Codificando el frontend

Para empezar, nos faltan un **componente**. Para ello creamos una carpeta dentro de la carpeta **app** llamada **components** y dentro un archivo **app.vue** la función del frontend empezará ahí, en **app**.

También es importante **añadir a index.html** (ver arriba el código index.html) **la referencia a app en un div**.



Antes de seguir, según el IDE a utilizar se puede instalar **vetur**, que es un complemento que **facilita el autocompletado** y resaltado de código de ficheros vue. En Visual Code probablemente te lo pedirá mediante un warning

Sigamos, vamos ahora a codificar un template sencillo en App.vu y modificar el index.js para ver que el frontend funciona.

App.vue



index.js



Compilamos:

```

C:\Users\user> cd E:\WebstormProjects\mevn
E:\WebstormProjects\mevn> npm run webpack

```

atch

```

d for emit] (name: main)
les

/ 432 KiB
es/@vue/ 431 KiB
e-dom/dist/runtime-dom.esm-bundler.js 58.9 KiB [built] [code genera
e-core/dist/runtime-core.esm-bundler.js 308 KiB [built] [code gener
vity/dist/reactivity.esm-bundler.js 41.4 KiB [built] [code generate

```

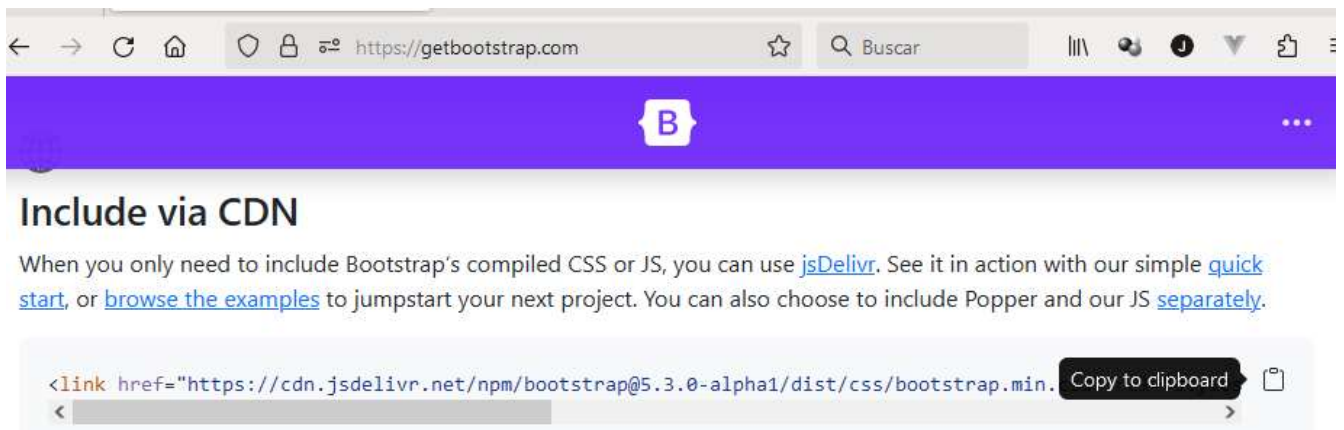
Y en el navegador:



Curso MEVN

Vamos a continuación a ir introducción algún estilo al frontend, es decir, a la vista de la aplicación.

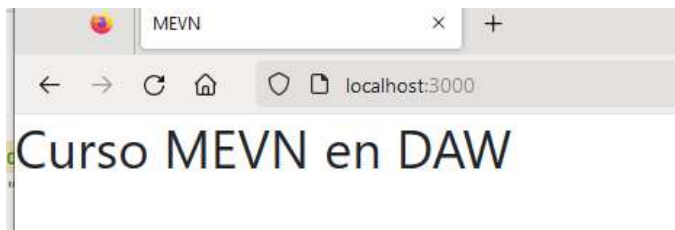
Para ello, vamos a introducir simplemente el **link a bootstrap** directamente desde CDN.



Y lo pegamos debajo del título.



Al reiniciar el servidor vemos que el título ya toma un estilo distinto



Vamos a crear poco a poco más contenido a la aplicación y empezamos con un menú. Para eso en **App.vue** codificamos lo siguiente, sin olvidarse de que hay que **compilarlo antes** de visualizar con el comando `npm run webpack` que ya quedará pendiente de nuevos cambios sin necesidad de volver a lanzarlo:

Código:



Compilación

```

./node_modules/babel-loader/lib/index.js!./node_modules/vue-loader/dist/templ
2)!./node_modules/vue-loader/dist/index.js??ruleSet[1].rules[5].use[0]!./src/app/
plate&id=b64e4c62 473 bytes [built] [code generated]
webpack 5.75.0 compiled successfully in 1936 ms

```

Resultado

Vamos a empezar con el método **POST** o alta de tareas

Nota.- Para que no haya problemas de compilación, en Vue es necesario que todo esté dentro de un **div**.

```

1  <template xmlns="http://www.w3.org/1999/html">
2    <div>
3      <nav class="navbar navbar-light bg-light">
4        <a href="/" class="navbar-brand">Inicio</a>
5      </nav>
6      <div class="container">
7        <div class="row pt-5">
8          <div class="col-mod-5">
9            <div class="card">
10             <div class="card-body">
11               <form>
12                 <div class="form-group">
13                   <input type="text" placeholder="Nueva Tarea"
14                     class="form-control">
15                 </div>
16                 <br>
17                 <div class="form-group">
18                   <textarea cols="30" rows="10"
19                     class="form-control" placeholder="Descripcion Tarea"></textarea>
20                 </div>
21                 <br>
22                 <div class="d-grid gap-2">
23                   <button class="btn btn-primary d-md-block">Guardar</button>
24                 </div>
25               </form>
26             </div>
27           </div>
28         </div>
29       </div>
30     </div>
31   </template>

```

Ya no hace falta compilar porque webpack al detectar cambios ya lo hace.

El resultado es:

Muy bien. Ahora vamos con la zona de script de **App.vue**. Pero antes modificamos algo de la plantilla añadiendo un método **altaTarea** y el modelo de los datos, como hicimos en las prácticas anteriores.

Nota. el método **prevent** evita que se recargue la página cada vez que hagamos **submit**.

Y el script, ya está comentado. Si nos fijamos usamos clases. Esto es así para solo tener que escribir el modelo una vez y no repetirlos campos cada vez que tengamos que instanciarlos en **data**, que con instanciar una nueva tarea **new tarea**, es suficiente.

Pero nosotros queremos **enviarle los datos al servidor**, no que los muestre por consola.

Vamos a modificar ligeramente las rutas en el fichero **index.js** concretamente en el método **app.use**, de la siguiente forma, para separar en el futuro nuevas aplicaciones desde el mismo servidor.

Luego nos vamos al fichero App.vue y como hicimos en APIREST usaremos los distintos métodos GET, POST, DEL. PUT.

Empecemos por el **método POST**:

Si nos fijamos sigue la misma estructura que en el ejemplo de la práctica anterior. El método, el formato JSON y la **res** o respuesta del servidor.

Veamos,

Vemos las tareas creadas, y si vamos /api/tareas:

Vemos las diferentes tareas creadas.

Pero ahora vamos al **método GET** que llamaremos listarTareas. Veamos el código, que ya puedes sospechar como es:

Y si recargamos la aplicación, vemos que recibe los datos.

Nota.- si no se observan y no te da error, reinicia el servidor.

Por cierto, que no se nos olvide, **estamos guardando en una base MongoDB**.

Una vez que ya podemos guardar datos, lo próximo es poder mostrarlos en una **plantilla HTML**.

En primer lugar, añadimos **un array para almacenar las tareas en data** que luego le enviamos al **template**.

Además, con fetch hacemos **res** al servidor para que me envíe los datos, es decir, las tareas.

A continuación, siguiendo siempre el mismo protocolo, preparamos **el template**.

El resultado

Vamos a ver ahora el método **DELETE**.

Antes de ver como elimina una tarea vamos a añadir el siguiente código para entender como funciona la función.

En la **template** tal como hicimos en la aplicación de APIRest, a medida que recorremos el listado añadimos un botón de eliminar.

Antes de proceder a eliminar, vamos a ver como trabaja el método a través del **_id** que le aporta MongoDB. Si codificamos el método de la forma siguiente:

Como vemos le pasamos un parámetro que es el id que le dió MongoDB (ver imagen de Compass) al dar de alta la tarea. Ejecutamos y pulsamos el botón:

Sin embargo, el fin último es eliminar para eso modificamos el método de acuerdo a lo siguiente:

Ejecutamos y vamos a eliminar la tarea de *Recursos Humanos* que es la última.

Antes

Después

Vemos que aunque la tarea fue eliminada necesitamos que la página **se recargue** para actualizarse. Entonces añadimos en el método que refresque la página. Por cierto, al principio puede que sea

un poco lento, debido a la compilación, a la recarga del servidor y alguna configuración final que queda.

Llegados a este punto nos queda el último método, **PUT o actualizar**.

En primer lugar vamos a crear un botón, al igual que hicimos con eliminar y en otros ejercicios que vaya recorriendo la lista.

Mostramos el código en **template**, igual al de **delete**, para modificar un registro tenemos que conocer su **id**.

Modificar o edit es quizás el método más complejo porque **consta de dos partes**. En primer lugar tiene que hacer una búsqueda del registro a modificar, y luego de modificarlo, queda grabarlo.

Pues bien, para eso tenemos que ir al fichero **tareas.js** del directorio **routes** y **crear una nueva ruta**, que es una ruta de búsqueda del registro a cambiar. Es decir, algo así:

Si nos fijamos es un **get**, pero un get especial, ya que no recorre todos sino que **hace una búsqueda de uno solo**. Ahora que ya tenemos la tarea que queremos cambiar, debemos rellenar el formulario con el contenido de ellas para hacer las modificaciones que deseemos. Para ello vamos al método editTarea y codificamos lo siguiente:

Si pulsamos un botón de una tarea, vemos que se carga en el formulario.

Ahora nos queda la **segunda parte**, la que permite **actualizar los datos en MongoDB**.

En primer lugar, tenemos que cambiar la vista cuando pulsemos Editar, es decir, se modifique la vista del formulario con los datos a modificar para que cuando pulsemos Grabar o Actualizar, dichos datos modificados mediante un request actualice la base de datos, o si el usuario no se decide anular y volver al formulario inicial. Veamos el código.

Si observamos hemos añadido una **sentencia condicional v-if**, y como tal, necesita una **condición, en este caso que la variable editar sea true o false**.

Por defecto, será false, así que, en la carga de datos **declaramos la variable editar a false**. Pero si **pulsamos el botón se pondrá a true** y el formulario entonces, al recargarse la página, mostrará el botón de Actualizar

Al pulsar **llama a editaTarea y se pone a true**

Y nos mostraría esto:

Para grabar los nuevos datos podríamos, como hicimos en el ejercicio anterior, crear un nuevo método, pero en este caso para ser más eficientes, usaremos el método **altaTarea**, que a fin de cuentas hace lo mismo. Pero, en nuestro caso, **añadimos un condicional indicando que**, si sabemos el id, cambie la orden a utilizar por MongoDB. Mejor ver el código siguiente:

Si nos fijamos **aparece la variable *tareaEdita*id**. El nombre lo dice todo, es el id de la tarea que queremos modificar, y como ***this.editar*** lo pusimos a true se ejecuta el else, y con la id de la tarea, ya solo hace falta invocar el método PUT. Nos queda claro está, declarar esa nueva variable, y como no, en la **sección data**

Y también que al **pulsar Editar, esa variable se cargue con el id de la tarea**, es decir:

Por ejemplo, la reunión, que estaba a las 10 la pasamos a las 11

Por cierto, en MongoDB, los cambios realizados

Nos queda añadir el **botón de Cancelar** en el caso de que no queramos actualizar, tal como hicimos en la práctica anterior.

Para ello incluimos el **botón Cancelar** en el código en el que se muestra el botón Actualizar, y que simplemente llame al método ***listTareas()***

Comprobamos, primero **pulsamos Editar** y luego el **botón Cancelar** que vuelve al inicio.

Puesta en producción

Solo nos queda poner en producción la aplicación. Para ello solo hay que seguir **la documentación de Vue**.

Nos vamos a `package.json` y configuramos lo siguiente

Antes de ejecutar el comando que se indica a continuación, decir que su ejecución simplificará el fichero ***bundle.js*** el cual en estos momentos es un fichero pesado y, sobre todo, bastante confuso. Es cierto que su puesta en producción no mejorará su legibilidad pero sí reducirá su extensión y lo hará más eficiente. Vamos a la **terminal, paramos webpack en desarrollo y ejecutamos:**

`npm run build`

Ahora lo único que tenemos que hacer es tomar el contenido de la carpeta `public` y subirla al servidor. En nuestro caso sería el fichero **`index.html` y la carpeta `js`**, pero también podríamos tener una carpeta **`css` o `sass`, imágenes,...** y ya estaría funcionando.

Esto sería una introducción. Nos faltaría ver aspectos de Vue como Nuxt que permite también crear aplicaciones JS basadas en APIRest, ampliar el tema de validaciones, Vuex, animaciones y transicioens e Vue o escalar la aplicación más allá de un Single Page.

Última modificación: Mércores, 15 de Febreiro de 2023, 12:23

◀ 3. MEVN. MongoDB, Express, Vue y Node. Configuración del Servidor. (1)

Ir a...

5. Vue Router. ▶

Vostede accedeu como Raúl Arias Pérez (Saír)
DIW_22_23_Ord

Resumen da retención de datos
Obter a apli móbil