MANUAL DE USUARIO
BROWSERIFY

1. ¿Qué es Browserify?

Browserify es una herramienta de JavaScript de código abierto que permite a los desarrolladores escribir módulos estilo Node.js que se compilan para su uso en el navegador. Por eso, Browserify le permite usar require en el navegador, de la misma manera que lo usaría en Node.

2. Instalación

Primero, nos moveremos a la carpeta donde se encuentra nuestro servidor node utilizando el comando “cd” en nuestra consola.

A continuación, escribiremos el siguiente comando:

```
npm install browserify
```

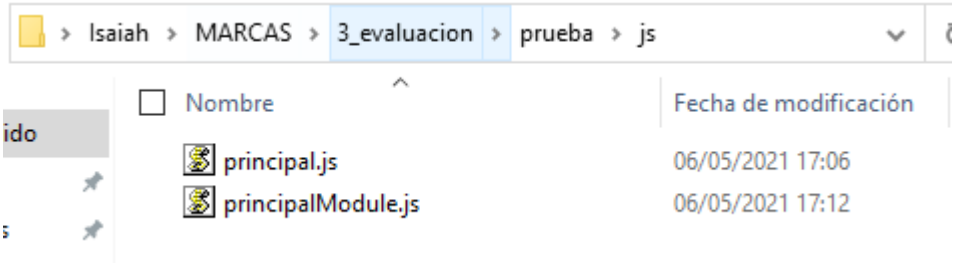
3. Cómo utilizar require()

Crearemos un paquete que exportará una función require() para que pueda utilizarse desde un módulo en tus archivos js.

```
browserify index.js --s module > dist/module.js
```

- “index.js” - Representa el js principal que utilizarás en tu servidor web.
- “module” - será la palabra que utilizaremos para hacer referencia al módulo en nuestros html. (Ejemplo:)
- “module.js” - será el encargado de exportar la función require() cada vez que se necesite.

De esta forma, obtendremos un nuevo .js en nuestro servidor (module), el cual no podremos editar a mano.



Por lo tanto, para hacer cualquier tipo de actualización en nuestro .js, tendremos que hacer los cambios necesarios en nuestro .js principal (index.js), y escribir el comando anterior actualizando nuestro “módulo.js” con los cambios realizados en el “index.js”.

El “módulo” utiliza su propia sintaxis (añadiendo una cabecera y pie de página), por lo que no podremos editarlo a mano, sino dejará de funcionar y no podremos utilizar require();

4. Acceder a JS (con require()) desde HTML

Simplemente escribiremos la siguiente etiqueta en el encabezado del documento html desde el cual queremos acceder a algún tipo de método.

< script src = " dist / module.js " > </ script >

De esta forma, podremos llamar a un método del js principal, llamando al módulo:

<input type="button" value="Aceptar" onclick="module.loginUsuario()" >

Como hemos visto, llamamos al método “loginUsuario()” localizado en “module”, mediante el atributo onclick=””; por lo tanto, se ejecutará al accionar sobre el botón.

Vemos cómo se ejecuta la función al hacer click en el botón.

```
(function(f){if(typeof exports==="object"&&typeof module!=="undefined"){module.exports=f()}else if(typeof
var numeroAleatorio = 0;

function miFuncion() {
    numeroAleatorio = Math.random();
}

function b() {

}

module.exports = {
    miFuncion:miFuncion,
    b:b
};
},{}]},{},[1])(1)
});
```

Comprobamos que se ha realizado:

```
> numeroAleatorio
< 0.8880480268994237
```

5. Todos los comandos básicos:

Syntax: browserify [entry files] {OPTIONS}

Standard Options:

--outfile, -o Write the browserify bundle to this file.
If unspecified, browserify prints to stdout.

--require, -r A module name or file to bundle.require()
Optionally use a colon separator to set the target.

```
  --entry, -e  An entry point of your app

 --ignore, -i  Replace a file with an empty stub. Files can be globs.

--exclude, -u  Omit a file from the output bundle. Files can be globs.

--external, -x  Reference a file from another bundle. Files can be globs.

--transform, -t  Use a transform module on top-level files.

 --command, -c  Use a transform command on top-level files.

--standalone -s  Generate a UMD bundle for the supplied export name.
                 This bundle works with other module systems and sets the name
                 given as a window global if no module system is found.

   --debug -d  Enable source maps that allow you to debug your files
               separately.

   --help, -h  Show this message

For advanced options, type `browserify --help advanced`.

Specify a parameter.
```