# Patrón Módulo

Es el patron de diseño mas comun que existe en Javascript ya que es compatible con casi todas las versiones. Permite encapsular y proteger el codigo.

Consiste en colocar nuestro codigo js dentro de una funcion anonima autoinvocable. Utilizar ‘use strict’

y lo que pongamos en el return de esta funcion va a ser publico.

## Modulos exportaciones por defecto e individuales

En el js que llama/usa las funciones que se exportan:

 import crearDeck, { miNombre } from './usecases/crear-deck';

En el js que estan definidas las funciones:

Individual

 export *const* miNombre = 'Fernando';

por defecto(colocar al final del archivo)

export default crearDeck;

# Mejorar la documentación JSDoc Comments

Instalar la extencion y luego con //\*\* nos deberia salir el atajo, le damos tab y nos debe rellenar de la siguiente manera:

/\*\*

 \*

 \* *@param* {\*} tiposDeCarta

 \* *@param* {\*} tiposEspeciales

 \* *@returns*

 \*/

Y colocar lo que creamos necesario

Ardenar selección:

A black rectangular object with a black stripe

Description automatically generated

# [javascript-minifier.com](https://javascript-minifier.com/)

Para ofuscar y minimizar nuestro codigo. Tambien existen extenciones en VsCode. Nos permite crear un archivo cuya nomenclatura deberia ser por ejemplo: juego-min.js y esto significaria que ya está listo para ser codigo productivo

# [Underscore.js (underscorejs.org)](https://underscorejs.org/)

Libreria js con varias funcionalidades:

# Clases

Viene a reemplazar el prototype y la manera de trabajar con funciones simulando las clases

## Sets, Gets y Propiedades, sets y metodos estaticos

*class* Persona {

    static \_conteo = 0;

    static *get* conteo() {

        return Persona.\_conteo + ' instancias';

    }

    static mensaje() {

        console.log( this.nombre ); // undefined

        console.log('Hola a todos, soy un método stático');

    }

    nombre = '';

    codigo = '';

    frase  = '';

    comida = '';

*constructor*( *nombre* = 'Sin nombre', *codigo* = 'Sin código', *frase* = 'Sin frase') {

        this.nombre = nombre;

        this.codigo = codigo;

        this.frase  = frase;

        Persona.\_conteo++;

    }

*set* setComidaFavorita( *comida* ) {

        this.comida = comida.toUpperCase();

    }

*get* getComidaFavorita() {

        return `La comida favorita de ${ this.nombre } es ${ this.comida }`;

    }

    quienSoy() {

        console.log(`Soy ${ this.nombre } y mi identidad es ${ this.codigo }`);

    }

    miFrase() {

        this.quienSoy();

        console.log(`${ this.codigo} dice: ${ this.frase }`);

    }

}

*const* spiderman = new Persona( 'Peter Parker', 'Spiderman', 'Soy tu amigable vecino Spiderman' );

*const* ironman   = new Persona( 'Tony Stark', 'Ironman', 'Yo soy Ironman' );

// console.log( ironman );

spiderman.miFrase();

// ironman.miFrase();

spiderman.setComidaFavorita = 'El pie de cereza de la tía May';

// spiderman.comida = 'Duende Verde';

// spiderman.nemesis = 'Duende Verde';// es es aceptado por JS y crea la propiedad aunque no esté en su definicion

// console.log( spiderman.getComidaFavorita );

// console.log( spiderman );

// Persona.\_conteo = 2;

console.log('Conteo stático', Persona.\_conteo );

console.log( Persona.conteo );

Persona.mensaje();

Persona.propiedadExterna = 'Hola Mundo';//esto es permitido pero no recomendado

console.log( Persona.propiedadExterna );

console.log( Persona );

Mas en ES6 y en el codigo fuente de JSModerno. Herencia, subclases

# Propiedades Privadas

Al momento de realizar el curso todavia no está globalmente aceptado por eso hay que revisar este link

[JavaScript classes: Private class fields | Can I use... Support tables for HTML5, CSS3, etc](https://caniuse.com/mdn-javascript_classes_private_class_fields)

Todavia no estan aceptados los metodos privados

*class* Rectangulo {

    #area = 0;

*constructor*(*base* = 0, *altura* = 0) {

        this.base   = base;

        this.altura = altura;

        this.#area = base \* altura;

    }

    calgularArea() {

        console.log( this.#area \* 2 );

    }

}

*const* rectangulo = new Rectangulo(10, 15);

// rectangulo.#area  = 100;

rectangulo.calgularArea();

console.log(rectangulo);

# Singleton

Es una instancia unica de una clase de manera global, util para compartir informacion a lo largo de la app, con BD, etc…

*class* Singleton {

    static instancia; // undefined

    nombre = '';

*constructor*( *nombre* = '' ) {

        if ( !!Singleton.instancia ) {

            return Singleton.instancia;

        }

        Singleton.instancia = this;

        this.nombre = nombre;

    }

}

*const* instancia1 = new Singleton('Ironman');

*const* instancia2 = new Singleton('Spiderman');

*const* instancia3 = new Singleton('BlackPanther');

console.log(`Nombre en la instancia1 es: ${ instancia1.nombre }`);

console.log(`Nombre en la instancia2 es: ${ instancia2.nombre }`);

console.log(`Nombre en la instancia3 es: ${ instancia3.nombre }`);

# Multiples Constructores

A diferencia de otros lenguajes JS no lo soporta pero se puede simular creando unmetodo estatico, es util siq queremos crear instancias de diferentes formas:

*class* Persona {

    static porObjeto({ *nombre*, *apellido*, *pais* }) {

        return new Persona( nombre, apellido, pais );

    }

*constructor*( *nombre*, *apellido*, *pais* ) {

        this.nombre   = nombre;

        this.apellido = apellido;

        this.pais     = pais;

    }

    getInfo() {

        console.log(`info: ${ this.nombre }, ${ this.apellido }, ${ this.pais }`);

    }

}

*const* nombre1   = 'Melissa',

      apellido1 = 'Flores',

      pais1     = 'Honduras';

*const* fher = {

    nombre:   'Fernando',

    apellido: 'Herrera',

    pais:     'Costa Rica'

}

*const* persona1 = new Persona( nombre1, apellido1, pais1 );

*const* persona2 = Persona.porObjeto( fher );

persona1.getInfo();

persona2.getInfo();

# Bundlers o empaquetador

Son empaquetadores de modulos. Nos ayudan a desarrollar una aplicación de JavaScript moderna, es decir, con módulos, separando archivos, que cada archivo tenga una responsabilidad única, que nosotros podamos importar diferentes tipos de tecnologías que podamos usar, sass o diferentes tipos de estilo de CSS y todo combinarlo, ejecutarlo, minimizar y ofuscar el codigo y crear un resultado final que es nuestra aplicación, entre comillas, lista para producción.

Genera el build de producción: tenemos archivos JavaScript, tenemos archivos de Handlebars, sass, png, ect. Se toman todos estos archivos, se pasan por el Bundler y genera el build de producción.

También hay otros pasos extras que podemos hacer, como por ejemplo pasarlo por Babbel, el cual nos permite a escribir código actual de JavaScript y transformarlo a código de cualquier versión que nosotros queramos de JavaScript. Tranformando un estandar a otro

## Webpack

[webpack](https://webpack.js.org/)

Webpack es un bundler muy popular que se utilizó por muchos años, ademas de todo lo nombrado anteriormente tambien tiene pluggins para que tengamos servidores en tiempo real, para que cuando se haga un cambio, mi aplicación automáticamente se recargue y se muestren los cambios en pantalla de nuevo. Gestiona dependencias, Monta servidores de desarrollo y pruebas. Uno de los principales problemas actuales de WP es su velocidad impide que podamos generar un código o desarrollar código más rápidamente. Su configuracion inicial no es simple y puede ser complicado detectar problemas cuando un paquete falla. React, Angular y Vue lo tienen incorporado por defecto

### Instalacion

[Getting Started | webpack](https://webpack.js.org/guides/getting-started/)

Dentro de la carpeta de desarrollo

npm install webpack webpack-cli --save-dev

Una vez instalado dentro del archivo package.json en la seccion scripts agregar una nueva linea: *"build": "webpack"*

*"scripts"*: {

*"test"*: "echo \"Error: no test specified\" && exit 1",

*"build"*: "webpack"

y ejecutar el siguiente comando en la terminal

*npm run build*

### Archivo de configuracion

[Configuration | webpack](https://webpack.js.org/configuration/)

[Mode | webpack](https://webpack.js.org/configuration/mode/)

Para configurar el ambiente en Desarrollo en la raiz de nuestro proyecto(No en el el SRC) crear un archivo llamado webpack.config.js y colocar lo siguiente:

*module*.*exports* = {

    mode: 'development',

}

### Pluggins

[html-loader | webpack](https://webpack.js.org/loaders/html-loader/)

[HtmlWebpackPlugin | webpack](https://webpack.js.org/plugins/html-webpack-plugin/)

### Crear Server local de desarrollo

[DevServer | webpack](https://webpack.js.org/configuration/dev-server/)

npm i -D webpack-dev-server

Una vez instalado dentro del archivo package.json en la seccion scripts agregar una nueva linea:

*"start"*: "webpack serve --config webpack.config.js --open --port=8080"

### Importando estilos de forma dinámica

[css-loader | webpack](https://webpack.js.org/loaders/css-loader/)

[style-loader | webpack](https://webpack.js.org/loaders/style-loader/)

Luego de realizar las instalaciones en el webpack.config.js agregar una nueva regla:

{

                test: /\.css$/,

                use: ["style-loader", "css-loader"]

            }

En plugins:

 new MiniCssExtract({

            filename: 'nuevo-estilo.css',

            ignoreOrder: false

        })

### Creando un archivo de estilos de forma global en la aplicación

[MiniCssExtractPlugin | webpack](https://webpack.js.org/plugins/mini-css-extract-plugin/)

Luego de instalar agregar una nueva regla en el archivo webpack.config.js para el archivo especifico:

  {

                test: '/styles.css$/',

                use: [MiniCssExtract.loader, 'css-loader']

            }

y colocar la siguiente linea en el inicio

const MiniCssExtract = require("mini-css-extract-plugin");

### Manejo de imágenes

[file-loader | webpack](https://v4.webpack.js.org/loaders/file-loader/)

Luego de la instalacion agregamos una nueva regla:

{

                test: /\.(png|jpe?g|gif)$/,

                loader: 'file-loader'

            }

### Mover recursos estaticos

[CopyWebpackPlugin | webpack](https://webpack.js.org/plugins/copy-webpack-plugin/)

Luego de instalar copiamos la siguiente linea en el config

const CopyPlugin = require("copy-webpack-plugin");

y crear el plugin

 new CopyPlugin({

            patterns: [

                { from: 'src/assets/', to: 'assets/' }

              ]

        })

## Webpack - Production Mode

copiamos el webpack.config.js y lo renombramos a webpack.prod.js y le cambiamos el

mode: 'production',

A screenshot of a computer

Description automatically generated

en los scripts del package.json modificamos los build

*"build"*: "webpack --config webpack.prod.js",

*"build:dev"*: "webpack --config webpack.config.js",

y compilamos con npm run build

***Ver codigo git para revisar todas las cofiguraciones***

## Instalación y configuración de Babel

Para que nuestra app pueda ser usada en todos los navegadores web Babel traduce nuestro codigo para que sea compatible

[Babel · Babel (babeljs.io)](https://babeljs.io/)

Instalacion: <https://babeljs.io/setup> pestaña Webpack y copiar el comando y la regla que nos indica en la pagina

Tal como indica la pagina crear el archivo babel.config.json en la raiz de nuestra app, instalar el preset con el comando y colocar el codigo que indica

## Node

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) asíncrono.

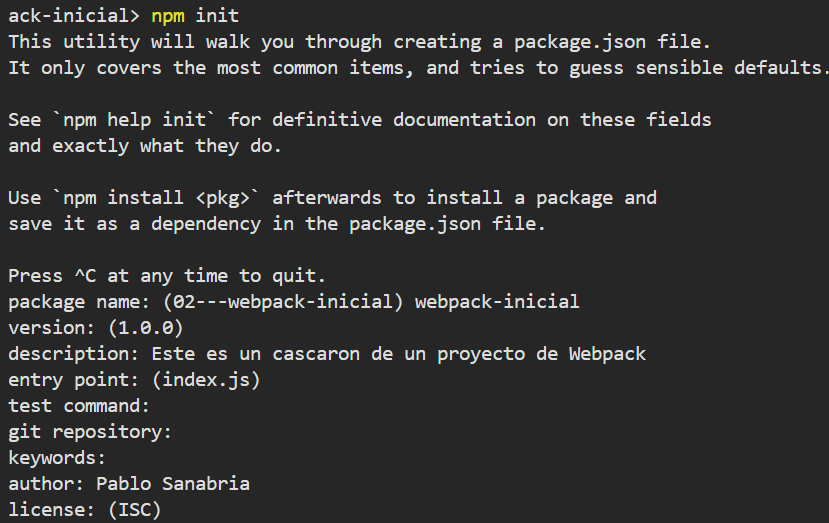
### NPM

Es el sistema de gestión de paquetes por defecto para Node

#### Crear el package.json

el cual es primordial y va a decir cómo funciona la aplicación. Nos dice que paquetes son necesarios para que la app funcione. Qué dependencias son necesarias para pasar a producción qué cosas tengo que descartar que cosas que sólo necesito en desarrollo, etc.

*npm init*

**

*A screen shot of a computer

Description automatically generated*

## Vite

Es una herramienta Similar a Webpack pero con mas velocidad. Ademas ya viene con un montón de código pre generado para permitir rápidamente no tener que hacer toda la configuración manual que tendríamos que hacer con Webpack

### Instalacion

Ejecutar el comando: *npm create vite* pregunta si queremos instalarlo colocar si(y) (con *npm create vite@latest* nos aseguramos que se cree con la ultima version)

Esto crea el cascaron inicial de un proyecto lo que antes debiamos hacer nosotros manualmente con Webpack

Colocamos el nombre del proyecto que queramos y nos da a alegir que framework queremos utilizar

A screen shot of a computer

Description automatically generated

Una vez seleccionado ejecutar lo que nos pide:

A black screen with white text

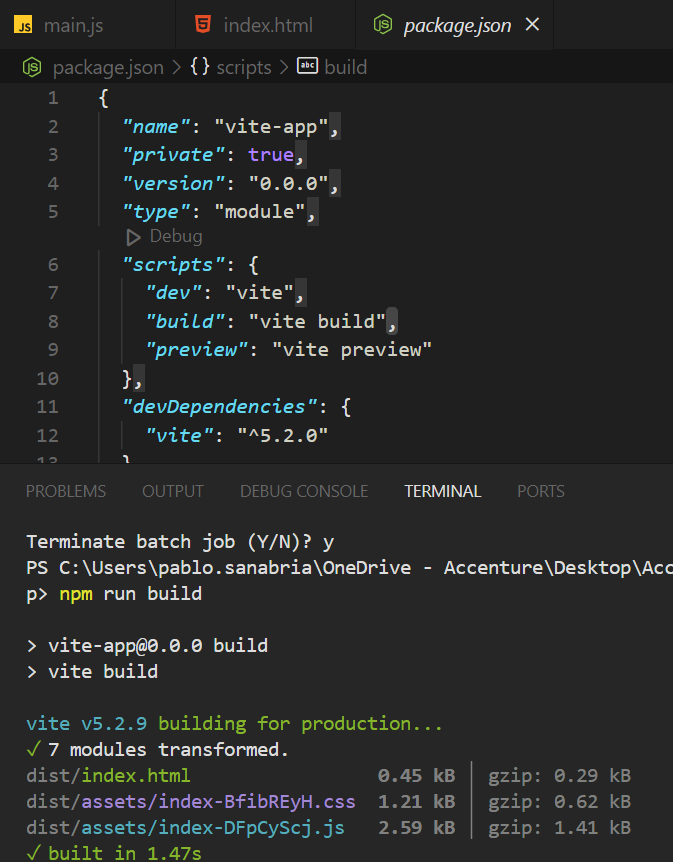
Description automatically generated

Con esto ya deberia levantar el ambiente y luego copiamos la url que nos aparece en un navegador

A computer screen shot of a program

Description automatically generated

### Generar build de Produccion



aparece la carpeta Dist(distribucion)

A black screen with white text

Description automatically generated

### Despliegue en la nube

#### <https://www.netlify.com/>

Es un servicio de Internet que tiene un tip gratuito que nos sirve para desplegar aplicaciones

rápidamente que únicamente dependan de código de JavaScript, CSS y HTML y que no tenga ningún código Back.

En la solapa Sites arrastrar la carpeta Dist

A screenshot of a computer

Description automatically generated

y listo, luego podemos cambiarle el nombre

#### Carpeta public

Aquí debemos colocar los recursos estaticos de nuestro proyecto que no van a cambiar durante el proceso de construccion. Por ende vite no le coloca hash a los nombres de los archivos

#### Desplegar en GitHub Pages

Renombrar nuestra carpeta dist con el nombre docs que va a contener la version final del proyecto

Para repositorios privados solo se puede realizar con una version paga, asi que deberiamos tener nuestro repo en public si queremos hacerlo gratis.

En la seccion Pages de Github -> deploy from a branch -> seleccionamos la rama y la carpeta docs y le damos save

A screenshot of a chat

Description automatically generated

Si no desplegamos nuestro proyecto en el route debemos configurar el route por defecto creando un archivo de configuracion de Vite,que en muchos frameworks se llama *vite.config.js* y volver a compilar con *npm run build* y volver a realizar el cambio de nombre de la carpeta dist.

## A screen shot of a computer Description automatically generated

Una vez que subimos la nueva carpeta docs con esta config, automaticamente nuestra page en github deberia recompilarse.

## TurboPack

Es el sucesor de Webpack aunque todavia está en pruebas (escrito en RUST)

# Store

Cuando los componentes o piezas de nuestra app necesiten informacion, en vez de pedirsela a otro componente o pieza puedo obtenerse del store, para esto es muy util separarlo en un archivo independiente

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## LocalStorage

Persiste durante todo el tiempo que viva la app, mientras no se formatee el servidor o el dispositivo siempre que estemos en el dominio indicado con el mismo usuario y el sistema operativo.

Hay mas espacio de almacenamiento que en las Cookies

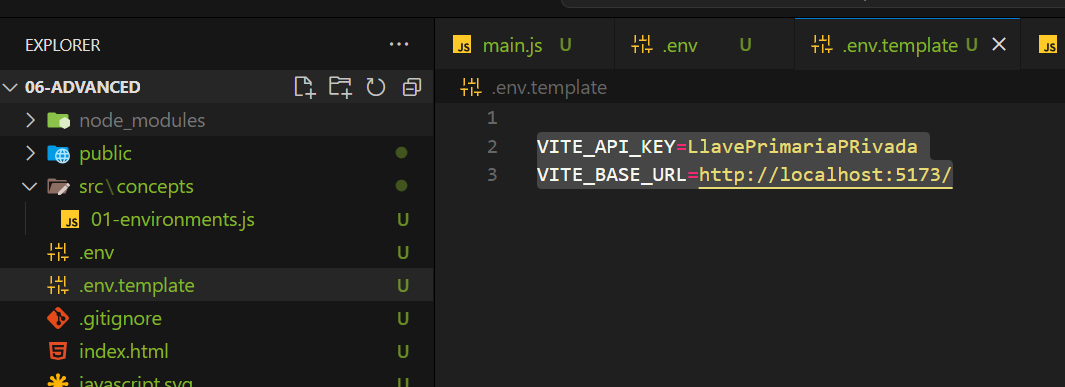
## SessionStorage

Se pierde cuando se reinicia la computadora , el navegador o el servidor.

# Variables de Entorno

Son variables que dependiendo de su entorno/ambiente(dev,int,homo,prod) o en donde se esten ejecutando van a tener un valor distinto. El paquete mas conocido es dotenv.

Es un estandar que esten en un archivo .env (de envariment variables). Se debe agregar en el gitignore para no darle seguimiento. Y se puede crear un .env.template para hacerlo de manera local



Vite por defecto tiene su forma de trabajar con estas variables:

console.log(import.meta.env);

A screenshot of a computer

Description automatically generated

# Callbacks

Es una funcion que recibe como argumento otra funcion que es invocada dentro de la primera funcion:

/\*\*

 \*

 \* *@param* {HTMLDivElement} element

 \*/

export *const* callbacksComponent = (*element*)*=>*{

*const* id = '5d86371f1efebc31def272e2';

    findHero(id, (*heroe*)*=>*{

        element.innerHTML = heroe.name;

    });

}

/\*\*

 \*

 \* *@param* {String} heroId

 \* *@param* {(heroe:Object)=>void} callback

 \*/

*const* findHero = (*heroId*, *callback*) *=>*{

    //heroes es una lista(json) invocada

*const* heroeCallback = heroes.find( *hero* *=>* hero.id === heroId);

    callback(heroeCallback);

}

## Manejo de errores

export *const* callbacksComponent = (*element*)*=>*{

*const* id = '5d86371f1efebc31def272e2';

    findHero(id, (*error*, *heroe*)*=>*{

        if(error)

        {

            element.innerHTML = error;

            return;

        }

        element.innerHTML = heroe.name;

    });

}

/\*\*

 \*

 \* *@param* {String} heroId

 \* *@param* {(error: String|null, heroe:Object)=>void} callback

 \*/

*const* findHero = (*heroId*, *callback*) *=>*{

    //heroes es una lista(json) invocada

*const* heroeCallback = heroes.find( *hero* *=>* hero.id === heroId);

    if(!heroeCallback){

        callback(`Heroe with id ${heroId} doesn´t exist`);

        return;

    }

    callback(null, heroeCallback);

}

## Callback Hell

Un Hell es cuando tenemos en nuestro codigo comportamientos, que se anidan, creando una dependencia de referencias eterna, en este caso de una funcion que es llamada por otra funcion.

# Promesas

/\*\*

 \*

 \* *@param* {String} id

 \* *@returns* {Promise<Object>}

 \*/

*const* findHero = (*id*)*=>*{

    return new *Promise*( (*resolve*, *reject*)*=>*{

*const* hero = heroes.find(*hero=>*hero.id === id);

        if (hero) {

            resolve(hero);

            return;

        }

        reject(`Hero with id ${ id } not found`);

    });

}

export *const* promiseComponent = (*element*)*=>*{

*const* renderHero = (*hero*)*=>*{

        element.innerHTML = hero.name;

    }

*const* renderError = (*error*)*=>*{

        element.innerHTML = error;

    }

*const* id1 = '5d86371f25a058e5b1c8a65e';

*const* id2 = '5d86371f2343e37870b91ef1';

    //findHero(id1).then(superHero => renderHero(superHero));

    findHero(id1).then(renderHero)//mejora de codigo

    .catch(renderError);

}

//Callback hell o promise hell

    findHero(id1)

        .then( (*hero1*)*=>*{

            findHero(id2).

                then(*hero2* *=>* renderTwoHeros(hero1,hero2))

                .catch(renderError);

        })

        .catch(renderError);

//cadena de promesas(con dependencia de resultados, para ejecutar una necesito el resultado de la otra)

*let* hero1;

    findHero(id1)

    .then(*hero=>*{

        hero1 = hero;

        return findHero(id2);

    }).then(*hero2=>*{

        renderTwoHeros(hero1,hero2);

    })

    .catch(renderError);

## Promise.All([])

Nos permite ejecutar todas las promesas definidas en un arreglo de promesas. Se puede realizar si las promesas no dependen entre si de sus resultados. Se ejecutan simultanemante, si una da error no se ejecuta el then:

*Promise*.all([

        findHero(id1),

        findHero(id2),

    ])

    .then(([*hero1*,*hero2*])*=>*renderTwoHeros(hero1,hero2))

    .catch(renderError);

## Promise.race([])

Es util cuando queremos devolver el resultado de la promesa que se resuelva mas rapido

export *const* promiseRaceComponent = (*element*)*=>*{

*const* renderValue = (*value*)*=>*{

        element.innerHTML  = value;

    }

*Promise*.race([

        slowPromise(),

        fastPromise(),

        mediumPromise(),

        fastFastPromise(),

        slowPromise()

    ])

    .then(renderValue)

    .catch('Error en el Race');

}

*const* slowPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Slow Promise`);

    }, 2000);

})

*const* mediumPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Medium Promise`);

    }, 1500);

})

*const* fastPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Fast Promise`);

    }, 1000);

})

*const* fastFastPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Fast Fast Promise`);

    }, 500);

})

# Funciones asincronas - Async

Las funciones async retornan siempre una promesa. No siguen el hilo principal(secuencial) porque javascript a las funciones async o promesas las coloca en una pila de tareas por realizar y se ejecuta a destiempo. Tener cuidado con los try y catch, ya que el catch dentro de una funcion asincrona se retorna como estado de éxito.

## Await

Espera a que el resultado de una funcion termine y cuando termina retorna el producto de esa promesa

export *const* asyncAwaitComponent = async (*element*)*=>*{

*const* id1 = '5d86371f97c29d020f1e1f6d';

*const* id2 = '5d86371fd55e2e2a30fe1ccb';

    try {

*const* hero1 = await findHero(id1);

*const* hero2 = await findHero(id2);

        element.innerHTML = `${hero1.name} / ${hero2.name}`

    } catch (error) {

        element.innerHTML = error;

    }

}

*const* findHero = async ( *id* ) *=>* {

*const* hero = heroes.find( *hero* *=>* hero.id === id);

    if(!hero)

        throw `Hero with id ${ id } not found`;

    return hero;

}

## Optimizar promesas no secuenciales

Cuando las promesas que no dependen una de otra, son independientes. Con Promise.all() se pueden ejecutar todas las promesas de manera simultanea teniendo mas performance y velocidad.

export *const* asyncAwait2Component = async (*element*)*=>*{

    try {

        console.time('Start');

        //Esto demora mas

    //    const value1 = await mediumPromise();

    //    const value2 = await fastPromise();

    //    const value3 = await fastFastPromise();

    //Esto demora menos

*const* [value1, value2, value3] = await *Promise*.all([

        mediumPromise(),

        fastPromise(),

        fastFastPromise()

    ]);

       element.innerHTML = `

       1- ${ value1 } </br>

       2- ${ value2 } </br>

       3- ${ value3 } </br>

       `;

       console.timeEnd('Start');

    } catch (error) {

        element.innerHTML = error;

    }

}

*const* mediumPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Medium Promise`);

    }, 1500);

})

*const* fastPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Fast Promise`);

    }, 1000);

})

*const* fastFastPromise = ()*=>* new *Promise*( *resolve* *=>* {

    setTimeout(() *=>* {

        resolve(`Fast Fast Promise`);

    }, 500);

})

## For await

Sirve cuando queremos recorrer un arreglo de promesas y que se resuelva cada una

export *const* forAwaitComponent = async( *element* ) *=>* {

*const* id1 = '5d86371f97c29d020f1e1f6d';

*const* heroIds = heroes.map(*hero* *=>* hero.id);

*const* heroPromises = getHeroesAsync(heroIds);

    for await ( *const* hero of heroPromises)

    {

        element.innerHTML += ` ${ hero.name } <br/>`

    }

}

/\*\*

 \*

 \* *@param* {Array<String>} heroIds

 \* *@returns* {Array<Promise>}

 \*/

*const* getHeroesAsync = ( *heroIds* ) *=>* {

*const* heroPromises = [];

    heroIds.forEach( *id* *=>* {

        heroPromises.push( getHeroAsync(id)  );

    });

    return heroPromises;

}

*const* getHeroAsync = async(*id*) *=>* {

    await new *Promise*(( *resolve* ) *=>* {

        setTimeout(() *=>* resolve(), 1000)

    });

    return heroes.find( *hero* *=>* hero.id === id );

}

# Funciones generadoras

Generan una secuencia de valores. No se puede usar con funcion de flecha. que devuelve un objeto Generator

export *const* generatoFuncComponent = (*element*)*=>*{

*const* generator = myFirstGeneratorFunction();

    console.log(generator.next());// valor 1

    console.log(generator.next());// valor 2

    console.log(generator.next());// valor 3

    console.log(generator.next());// ultimo valor

    console.log(generator.next());// undefined

    console.log(generator.next());// undefined

}

*function*\* myFirstGeneratorFunction() {

    yield 'valor 1';

    yield 'valor 2';

    yield 'valor 3';

    return 'Ultimo valor';

}

A screenshot of a computer code

Description automatically generated

Uso comun generar ids secuenciales:

export *const* generatoFuncComponent = (*element*)*=>*{

*const* genId = idGenerator();

*const* button = document.createElement('button');

    button.innerText = 'Click me';

    element.append( button );

*const* renderButton = ()*=>* {

*const* {value} = genId.next();

        button.innerText = `Click numero ${value}`;

    }

    button.addEventListener('click', renderButton);

}

*function*\* idGenerator() {

*let* currentId = 0;

    while(true)

        yield ++currentId;

}

## Async

export *const* generatorAsync = async (*element*)*=>*{

*const* heroGenerator = getHeroGenerator();

*let* isFinished = false;

    do {

*const* { value, done } = await heroGenerator.next();

        isFinished = done;

        element.innerText = value;

    } while (!isFinished);

}

async *function*\* getHeroGenerator() {

    for(*const* hero of heroes){

        await sleep();

        yield hero.name;

    }

    return 'No hay mas heroes';

}

*const* sleep = ()*=>*{

    return new *Promise*((*resolve*)*=>*{

        setTimeout(() *=>* {

            resolve();

        }, 500);

    });

}

# Peticiones HTPP

## [Fetch API - Web APIs](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)

1. Operaciones CRUD
   1. Create
   2. Read
   3. Update
   4. Delete
2. Sin frameworks o librerías externas
3. Modals
4. Mappers
5. Clases
6. Utilizar conversiones
7. Código limpio
8. Comprender las funcionalidades básicas de un Restful API de un backend
9. Montar un backend de pruebas
10. Generar y utilizar un store central para nuestro estado de la aplicación
11. Documentación de funciones

## ATAJOS

CTRL + } = Comenta y descomenta (en mi teclado logitech)

F2 : luego se seleccionar una palabra(variable) podemos modificarla en todos los lugares al mismo tiempo

Para comentar funciones, luego de definirla, arriba de la funcion presionar /\*\* + tab: genera la estructura de comentario de funciones

Para importaciones presionar imp + tab