Diseño instruccional

**Unidad 2: Creando una aplicacióncon Javascript**

**Sugerencias generales**:

* Al inicio de la presentación para la clase exponer la tabla de contenido de la misma, de esta manera por ejemplo:



Tema 1

Tema 2

Tema 3

Tema 4

Tema 1: Asincronismo en Javascript

Tema 2: Introducción a SPA y PWA

Tema 3: Router en JS

Tema 4: Creando SPA con rutas

Tema 5: Despliegue SPA en github Pages

Tema 6: Debugging

Tema 7: Introducción a ReactJS

Tema 8: Instalar CRA (create react app)

Contenido de la clase

* Al iniciar cada tema: (durante este documento se indica cuándo emplearlo)



Nombre del tema

Idea general de él

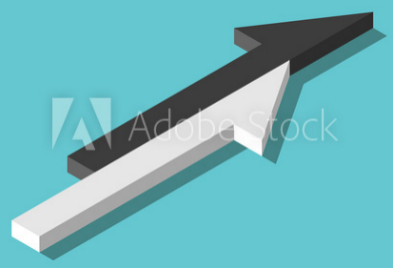
**TEMA 01**

* Si es posible, a medida que el docente expone, que la información vaya apareciendo con transiciones para guiar la lectura visual del estudiante y lograr mayor interiorización.
* Resaltar palabras claves. Se dejan aquí en negrilla.
* Durante este documento se reubicó la información como se sugiere ser presentado.

**TEMA 1**

**Asincronismo en Javascript**

**Asincronismo**

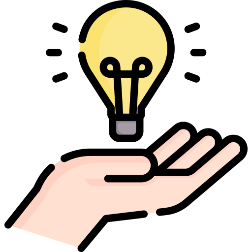


JavaScript es un lenguaje de programación asíncrono. Lo que quiere decir esto es que al ejecutar código JavaScript el hilo de ejecución continuará a pesar de encontrarse en situaciones en las que no obtenga un resultado inmediatamente.

|  |  |
| --- | --- |
| Por ejemplo, cuando hacemos el pedido de información a un servidor, la respuesta posiblemente demore un poco. Sin embargo, el hilo de ejecución de JavaScript continuará con las demás tareas que hay en el código.  Un ejemplo práctico de esto sería una aplicación web que necesita llenar una tabla de datos, así que el código hará un pedido al servidor de los datos que necesita llenar. Pero el hilo de ejecución no se detiene así que ejecutará el código que pinta la tabla en el navegador. | C:\Users\DORIAN GARZON\Desktop\CU01103E_2.jpg |



Esto se convierte en un problema ya que los datos del servidor llegan después de que la tabla se haya pintado en pantalla, una tabla sin datos obviamente.



Así que trabajar con código asíncrono puede tener muchas ventajas pero en casos como este, presenta un gran problema. Pues bien, para solucionar esto algunas funciones de JavaScript tienen como parámetro algo que se conoce como callback, que estudiaremos a continuación.

**Callbacks**

(Devolución de llamada o **retrollamada)**

|  |  |
| --- | --- |
| Es la primera y la forma más común de controlar la asincronía en JavaScript. En el siguiente ejemplo tenemos una función que recibe como parámetros un dato de entrada: |  |

La actividad de la función es muy simple:

Y cuando termine, llama a la función de callback que recibe por parámetro

Al array se le añade el data que viene por parámetro

En ese caso la llama con el array modificado.

Se ha añadido un pequeño bloque para comprobar si el array existe y si no lanzar un error que pasaremos al callback.

1function addToArray (data, array, callback) {

2 if (!array) {

3 callback(new Error('No existe el array), null)

4 } else {

5 array.push(data)

6 callback(null, array)

7 }

8}

En este código vemos cómo llamará a esta función y tratar el callback:

1var array = [1,2,3];

2

3addToArray(4, array, function (err) {

4 if (err) return console.log(err.message)

5 console.log(array)

6})

7

8// [1, 2, 3, 4]

|  |  |  |
| --- | --- | --- |
| **¿Esto que va a devolver?** | Plantilla de infografía de línea de tiempo degradada vector gratuito  Y nos mostrará el array con el nuevo dato  Se ejecutará el callback  Cuando se termine de ejecutar la función addToArray | Parece una tontería porque podríamos añadir el dato al array y después imprimirlo con console.log, pero imaginemos que esa operación de añadir un item al array fuera asincrónica, como podría ser una llamada vía AJAX. |

Para simular esto vamos a utilizar la función **setTimeout** para añadir un retardo de 1 segundo:

1function addToArray (data, array, callback) {

2 if (!array) {

3 return callback(new Error('No existe el array', null)

4 }

5 setTimeout(function() {

6 array.push(data)

7 callback(null, array)

8 }, 1000)

9}

10

11var array = [1,2,3];

12

13addToArray(4, array, function (err) {

14 if (err) return console.log(err.message)

15 console.log(array)

16})

17// (1 seg de delay)-> [1, 2, 3, .

**TEMA 2**

Introducción a SPA y PWA

**Aplicaciones Web Progresivas**

|  |  |  |
| --- | --- | --- |
| **PWAs**  «Progressive Web Apps» | Las aplicaciones web progresivas son aplicaciones web que utilizan APIs y funciones emergentes del navegador web junto a una estrategia tradicional de mejora progresiva para ofrecer una aplicación nativa —como la experiencia del usuario para aplicaciones web multiplataforma. | Qué son las aplicaciones web progresivas o PWA?: Breve guía para  entenderlas - TECNOVA |

Las aplicaciones web progresivas son un patrón de diseño útil, aunque no son un estándar formalizado. Se puede pensar que PWA es similar a AJAX u otros patrones similares que abarcan un conjunto de atributos de aplicación, incluido el uso de tecnologías y técnicas web específicas.

Para poder llamar PWA a una aplicación web, técnicamente hablando debe tener las siguientes características:

Imagen que contiene tabla, computer, mujer, computadora

Descripción generada automáticamente

Contexto seguro (en-US) ([HTTPS](https://developer.mozilla.org/es/docs/Glossary/https))

Archivo de manifiesto

Uno o más Servicio Workers

|  |  |  |
| --- | --- | --- |
| [Contexto seguro (en-US)](https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts) ([HTTPS](https://developer.mozilla.org/es/docs/Glossary/https))  Cyber security  La aplicación web se debe servir a través de una red segura. Ser un sitio seguro no solo es una buena práctica, sino que también establece tu aplicación web como un sitio confiable, especialmente si los usuarios necesitan realizar transacciones seguras. La mayoría de las funciones relacionadas con una PWA, como la geolocalización e incluso los servicios workers, solamente están disponibles cuando la aplicación se ha cargado mediante HTTPS. | [Servicio workers](https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API)  Monitor  Un servicio worker es un script que permite interceptar y controlar cómo un navegador web maneja tus solicitudes de red y el almacenamiento en caché de activos. Con los servicios worker, los desarrolladores web pueden crear páginas web rápidas y fiables junto con experiencias fuera de línea. | [El archivo manifest](https://developer.mozilla.org/es/docs/Web/Manifest)  User interface free icon  Un archivo [JSON](https://developer.mozilla.org/es/docs/Glossary/JSON) que controla cómo se muestra tu aplicación al usuario y garantiza que las aplicaciones web progresivas sean detectables. Describe el nombre de la aplicación, la URL de inicio, los iconos y todos los demás detalles necesarios para transformar el sitio web en un formato similar al de una aplicación. |

**Single Page Application**

|  |  |
| --- | --- |
|  | Web free iconUna **SPA** o Aplicación de una Sola Página es una web que todo su contenido carga una sola vez, es decir, tu navegador descarga un sólo archivo html con todos los recursos y dependencias que necesite para funcionar: estilos, scripts, imágenes, fuentes, etc. |
| Virtual free iconUna vez que carga todo el contenido de la aplicación la navegación de esta suele ser muy rápida y fluida, pues el contenido ya ha sido previamente cargado y sólo estamos intercambiando contenido a través de las interacciones del usuario. |
| Esto se hace mediante peticiones asíncronas con AJAX de tal manera que sólo se tiene que cargar el nuevo contenido.  Dependiendo de la complejidad y robustez de la aplicación también podría cargarse contenido de forma dinámica sin la necesidad de recargar la página. |

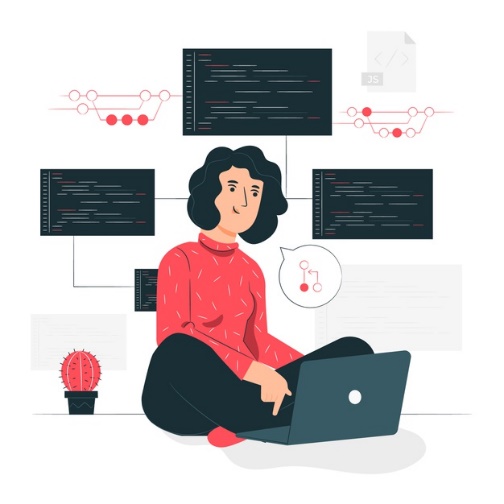
**Rutas en una SPA**

|  |  |  |
| --- | --- | --- |
| **En una SPA tenemos vistas, NO páginas**.  Eye | Las vistas son las diferentes secciones o apartados de contenido que tiene la aplicación y al no tener que cambiar de ruta para acceder al contenido, **la carga suele ser muy rápida.** |  |

¿Entonces si el contenido se carga en una sola página, la url no cambia?

Si... y No... 🤔😵😕.

|  |  |
| --- | --- |
| Por **funcionalidad** quizás **NO** sería necesario que cambiase la url | Pero por **usabilidad** es una muy buena opción que **SI** cambie |



Ya que el navegador va guardando en su historial todas las páginas (en este caso todas las vistas) que el usuario va visitando, lo que podría permitir al usuario regresar o adelantar vistas a través de los botones de atrás y adelante de su navegador web.

También piensa en la posibilidad que existe de que el usuario en vez de navegar por la aplicación para llegar a un contenido en particular, decida acceder a este a través de una url que decida **escribir manualmente en la barra de dirección de su navegador web.**

Si no implementas un sistema de rutas esto podría ser un gran problema de accesibilidad en el contenido de tu SPA.

|  |  |
| --- | --- |
| Programador trabajando en estilo isométrico vector gratuito  ¿En qué se programa una SPA? | Las SPAs se desarrollan en **JavaScript**, como son aplicaciones web 100% hechas con código **frontend**, es el único lenguaje en el que se pueden programar. También necesitaremos HTML para el marcado del contenido y CSS para su presentación. |

**TEMA 3**

**Router en JS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Getting Started with ReactJS - Knoldus React Blog | | | A continuación veremos el manejo de rutas del lado del **front-end** con una librería inspirada en **express router** donde se podrá ver unos ejemplos de su implementación de toda la [documentación](https://github.com/visionmedia/page.js) de esta librería. | Esta librería permite:   * Manejar las rutas de nuestra app sin recargar la página, * Mantiene nuestro historial de navegación, * Podemos pasar datos vía get al igual que podemos usar middlewares para encadenar funciones. | |
| Html | Lo primero es descargar la librería, en este caso, se usará un **cdn** para usar la librería de **pagejs** todo esto montado en una pequeña app con [**nodejs y express**](https://blog.johnserrano.co/servidor-basico-con-node-js/). | | |

Una etiqueta p

Y una lista con unos enlaces

#index.html

<p id="message"></p>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/user">User</a></li>

<li><a href="/user/jandrey15">User Name</a></li>

<li><a href="/notfound">Not found</a></li>

<li><a href="/ergtgt">Cualquier cosa</a></li>

</ul>

Hacemos nuestro llamado a la librería

<script src="https://cdn.rawgit.com/visionmedia/page.js/master/page.js"></script>

Dentro del mismo html agregamos un script:

Y la ejecución de unas funciones, como un callback podemos escribir la función hay mismo o pasarla con el nombre dentro de la función.

Donde tenemos la ejecución de page indicando la ruta.

<script type="text/javascript">

page('/', index)

page('/user', user)

page('/user/:username', user)

page('\*', notfound)

page()

function index() {

document.getElementById('message')

.textContent = 'Viewing home'

}

function user(ctx) {

document.getElementById('message')

.textContent = `Viewing user

${ctx.params.username || ''}`

}

function notfound(ctx) {

document.getElementById('message')

.textContent = 'Viewing not found'

}

</script>

Lo que hacemos es:

|  |  |  |
| --- | --- | --- |
| Seleccionar el elemento **p** con el **id message** y **enviar un texto.** | De esa forma sabemos que **page** está funcionando. | La barra de la ruta cambia según donde dimos click de los enlaces del html. |

Otro ejemplo:

Pasando el **CTX** y **NEXT** para trabajar con middlewares, veamos cómo se trabaja un ejemplo muy sencillo para ver su funcionamiento y se pueda entender.

Función Nombre

Agregamos en el html un <li> mas con una ruta

Y le pasamos dos funciones, una “nombre” y otra “apellido”.

Agregamos esa ruta con “page”

<li><a href="/cambio">Cambiar nombre apellido</a></li>

*// Se agrega el li al html*

page('/cambio', nombre, apellido)

*// Se agrega al script*

function nombre(ctx, next) {

document.getElementById('message')

.textContent = 'Viewing john'

ctx.nombre = 'John'

next()

}

function apellido(ctx) {

setTimeout(() => {

document.getElementById('message')

.textContent = `Viewing ${ctx.nombre} Serrano`

}, 2000)

}

>

La primera como siempre, pasamos el **ctx** y ahora también podemos pasar **next**,

Agregamos un **textContent,**

Y hacemos un next() para que siga con la siguiente función.

Seguido de eso pasamos el **ctx.nombre** para poder usar ese ctx en la siguiente función

Función Apellido

Como se puede ver podemos pasar funciones como middlewares y también podemos pasar datos con el ctx entre funciones.

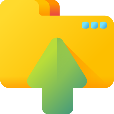
Y cambia el textContent de elemento p pasando el ctx.nombre

solo tenemos un setTimeout que se ejecuta pasado 2 segundos

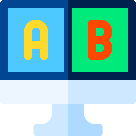
**TEMA 4**

**Creando SPA con rutas**

La lógica detrás de un enrutador contiene los siguientes pasos.

1. **Cargar la ruta**

Identificar dónde nos encontramos en el sitio. Se realiza a través de una carga inicial de la ruta.

1. **Comparar la URL con una de nuestras rutas**

La URL a la que nos queremos mover, se debe comparar con las rutas que tenemos ya que la ruta solicitada debe de estar entre nuestras rutas definidas para poder ser cargada.

1. Icono

   Descripción generada automáticamente**Actualizar la ruta en la barra de navegación**.

Para esto podemos utilizar un método de HTML conocido como pushState.

Este método forma parte del objeto wi

ndow de nuestro navegador windows.history.pushState. Este agrega un estado a nuestra historia de navegación que se refleja en la barra de navegación.

1. **Actualizar el DOM con el nuevo contenido**. El nuevo contenido se puede mandar a través de innerHTML.

**Arquitectura de nuestra aplicación**

|  |  |
| --- | --- |
| Hierarchical structure  Esta contempla un archivo HTML donde vivirá nuestro template el cual recibirá **el contenido** que va a estar cambiando en el DOM.  Interfaz de usuario gráfica, Aplicación, Word  Descripción generada automáticamente | Solution free icon  Por otro lado la **lógica de la aplicación** estará en nuestros archivos .js los cuales serán tres como se muestra en el siguiente diagrama.  Interfaz de usuario gráfica, Aplicación, Word  Descripción generada automáticamente |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |- root | |-- /js | |--- index.js | |--- router.js | |--- routes.js | |-- index.html | |-- style.css |



**¿Por qué SPA?**

Las SPA como su nombre lo indica son páginas web de una sola página y la navegación dentro de ella ocurre sin la necesidad de recargar el navegador.

💡 El propósito de realizar el routing con JavaScript vanilla es entender el funcionamiento de las librerías de routing de los frameworks.



Esta API trae un método llamado pushState que nos ayuda a traer los datos del stack de la navegación.



La cual no es una característica propia de JavaScript si no del navegador.



Para trabajar con la URL haremos uso de la API history

Interfaz de usuario gráfica

Descripción generada automáticamente

**Vamos al código**

|  |  |
| --- | --- |
| **1**  **Estructura** | Empezamos dándole estructura a nuestro template de HTML. Recordemos que la funcionalidad de esta pequeña práctica es hacer un router con JavaScript Vanilla por lo que nuestro template será lo más sencillo posible. |
| **2**  **Diccionario** | Con nuestro template HTML realizado el siguiente paso será definir nuestro diccionario de rutas, aquí estará una lista de todas las rutas disponibles dentro de nuestra SPA y el contenido de cada una de ellas. |
| **3**  **Router** | El archivo siguiente a trabajar será el que contiene toda la lógica para hacer funcionar nuestras rutas, si ahora trabajaremos en el router.js. En el cual definiremos una clase llamada Router, esta clase es la que inicializaremos en nuestro index.js, pero eso lo veremos más adelante.  class Router { |
| **4**  **Constructor** | Lo primero será definir un constructor para nuestra clase. Las funciones de este constructor será cargar nuestro listado de rutas e inicializar el router.  constructor(paths) {  this.paths = paths;  this.initRouter();  } |
| **5**  **Identificar ruta** | Con nuestro constructor creado escribiremos la lógica para iniciar el router. Esta función se encarga de identificar la ruta en la que se encuentra nuestro navegador.  initRouter() {  const { location: { pathname = "/" } } = window;  const URL = pathname === "/" ? "home" : pathname.replace("/", "");  this.load(URL);  } |
| **6**  **Cargar páginas** | Con la ruta identificada, podemos empezar a trabajar en la función que se encargará de cargar las páginas que desee visualizar el usuario. Esta será la función load que recibe por defecto el parámetro de home ya que queremos que ese contenido sea el contenido principal de nuestra SPA.  load(page = "home") {  const { paths } = this;  const { path, template } = paths[page] || paths.error;  const $CONTAINER = document.querySelector("#content");  $CONTAINER.innerHTML = template;  window.history.pushState({}, "done", path);  }  }  } |
| **7**  **Index.js** | Ahora con toda la lógica de nuestro router construida sólo nos resta inicializar la clase en el index.js.  const ROUTER = new Router(PATHS); |
| **8**  **Prueba** | Live Server - Visual Studio Marketplace  Ahora para poder probar correctamente nuestra pequeña SPA es necesario tener un servidor local. Esto se puede implementar de muchas maneras, pero yo recomiendo que si están usando VSCode instalen el **Live Server** el cual les hará la vida más fácil. |

**Conclusiones**

Así es como se trabaja los routers de algunos frameworks de JavaScript como **Angular o Vue**.

No es recomendable realizar "a mano" el router a menos que vaya a ser un proyecto muy pequeño.

Lo mejor es utilizar las implementaciones propias de cada framework.

Aunque siempre es bueno desglosar un poco las cosas y conocer desde las bases.



**Empezando con single-spa**

**JavaScript Microfrontends**

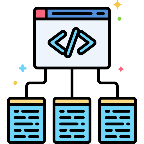
Single-spa es un framework que permite reunir múltiples microfrontends de JavaScript en una aplicación frontend. Crear la arquitectura de tu frontend usando single-spa te permite muchos beneficios, como:

Cargar el código con Lazy Loading, para mejorar el tiempo de carga inicial.

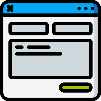


Usar múltiples frameworks en la misma página sin refrescar

(React, AngularJS, Angular, Ember, o cualquiera que estés usando)



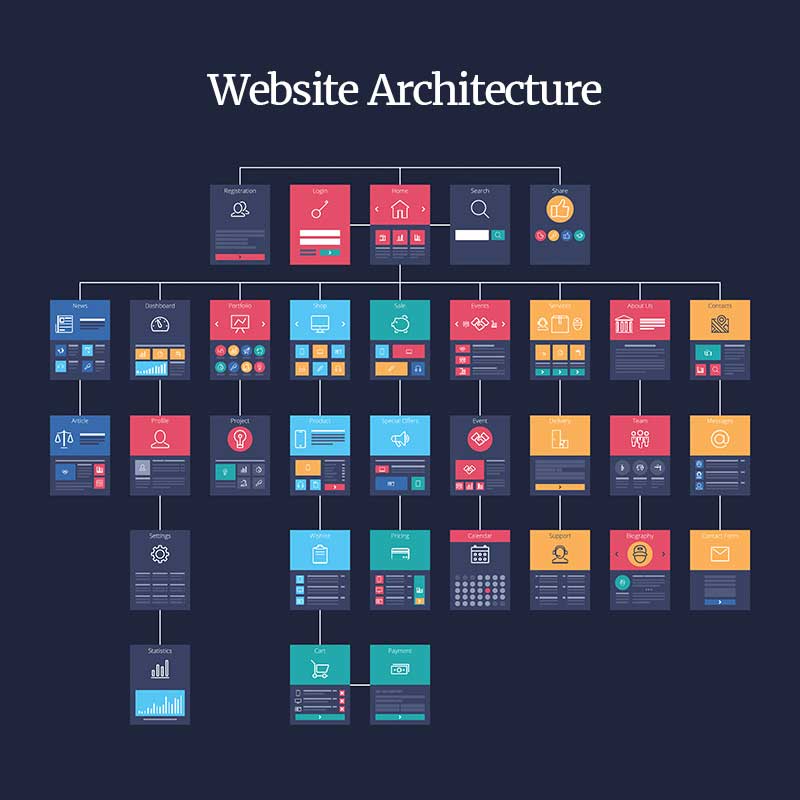
Desplegar tus microfrontends independientemente.



Escribir código usando un nuevo framework, sin tener que reescribir tu aplicación existente.



**Descripción de la Arquitectura**

Single-spa se inspira en el ciclo de vida de frameworks modernos, aplicando ciclos de vida a aplicaciones completas.

Nació del deseo de Canopy para utilizar React + react-router en lugar de quedarse con AngularJS + ui-router, y ahora single-spa soporta casi cualquier framework.

Dado que JavaScript es conocido por la corta vida de muchos de sus frameworks, se ha decidido facilitar su uso para cualquier framework que quieras.

Las aplicaciones de single-spa consisten en:

Idea 1

Cada aplicación puede responder a eventos de enrutamiento de una URL y sabe cómo arrancar, montarse y desmontarse así mismo desde el DOM.

Idea 2

La principal diferencia entre un **SPA** tradicional y las aplicaciones **single-spa** es que deben ser **capaces de coexistir** con otras aplicaciones, y no tienen su propia página HTML.

Idea 3

Por ejemplo, tu SPA de React o Angular son aplicaciones. Cuando se activan, escuchan los eventos de enrutamiento de la URL y colocan el contenido en el DOM. Cuando están inactivos, no escuchan los eventos de enrutamiento de la URL y se eliminan por completo del DOM.

El cuál es la página de HTML y Javascript que registra las aplicaciones con single-spa

**=**

Un single-spa-config

Cada aplicación es registrada con 3 cosas:

Una función que determina cuando la aplicación está activa/inactiva.

Una función para cargar el código de la aplicación.

Un nombre

**TEMA 5**

**Despliegue SPA en GitBub Pages**

|  |  |
| --- | --- |
| Deploy A React app on GitHub pages - DEV Community  Sitio "social coding" | Git/GitHub es un sistema de control de versiones muy popular, la mayoría de las empresas de tecnología ahora lo utilizan en su flujo de trabajo.  Permite subir repositorios de código para almacenarlo en el sistema de control de versiones Git. Puedes colaborar en proyectos de código, y el sistema es código abierto por defecto, lo que significa que cualquiera en el mundo puede encontrar tu código en GitHub, usarlo, aprender de él, y mejorarlo.  ¡Tú puedes hacer eso con el código de otras personas también! |

**https://www.youtube.com/watch?v=vCu7OKYeFlc**

Despliega tus aplicaciones a Github Pages

Para desplegar nuestros cambios a Github pages, vamos a utilizar el paquete angular-cli-ghpages

* Necesitas tener un usuario Github
* Necesitas crear un repositorio para tu proyecto.
* Necesitas guardar todos los cambios que haces en el proyecto en git
* Necesitas instalar angular-cli-ghpages
* Crear un usuario Github
* Si tu ya tienes un usuario Github puedes saltar este paso.

Para crear un usuario en Github ve a: https://github.com/ Llena el formulario de registro y asegúrate de validar tu dirección de correo electrónico.

Crea un repositorio para tu proyecto.

Después de acceder a Github.

Da click en Iniciar un proyecto y nombra el repositorio ng-girls-todo o cualquier nombre que quieras.

Conectando tu repositorio

Guarda todos los cambios en git ejecutando este comando en tu directorio de proyecto.

git add -A && git commit -m "Your Message"

Ejecuta el siguiente comando para conectar tu código al repositorio. Asegúrate de reemplazar {YOUR\_USERNAME} y {YOUR\_REPO} con tu nombre de usuario y nombre de repositorio.

git remote add origin https://github.com/{YOUR\_USERNAME}/{YOUR\_REPO}.git

git push -u origin master

Desplegar a Github pages

Primero instala angular-cli-ghpages

npm i -g angular-cli-ghpages

Luego solo ejecuta:

ng build --prod --base-href="/[your-repo-name]/"

angular-cli-ghpages

Fuente: <https://contraslash.github.io/todo-list-tutorial-spanish/deploy_to_githubpages.html>

**Preparando tu código para subirlo**

Puedes almacenar cualquier código que quieras en un repositorio de Github, pero para usar la característica páginas de Github con pleno efecto, debes:



Tu código debe estar estructurado como un sitio web típico, por ejemplo que el punto de entrada primario sea un archivo HTML llamado index.html.



Debes inicializar el directorio de código como un repositorio Git, para hacer esto:

1. Apunta la línea de comandos a tu directorio test-site (o como se llame el directorio que contiene tu sitio web). Para esto, usa el comando cd (Es decir "cambio de directorio"). Esto es lo que deberías digitar si has puesto tu sitio web en un directorio llamado test-site en tu escritorio.
2. Cuando la línea comandos está apuntando hacia el interior del directorio de tu sitio web, digita el siguiente comando, que le dice a la herramienta git para convertir el directorio en un repositorio git.
3. An aside on command line interfaces
4. La mejor manera de subir tu código a Github es mediante la linea de comandos.

**TEMA 6**

**Debugging**

Antes de escribir código más complejo, hablemos de debugging.

Todos los exploradores modernos y la mayoría de los otros ambientes soportan el “debugging” – una herramienta especial de UI para desarrolladores que nos **permite encontrar y reparar errores más fácilmente.**



Aquí utilizaremos Chrome porque es uno de los que mejores herramientas tienen en este aspecto.

**El panel “sources/recursos”**

Abre la página de ejemplo en Chrome.

Selecciona el panel sources/ recursos.

Activa las herramientas de desarrollo con F12 (Mac: Cmd+Opt+I).

Esto es lo que se debería ver si se está haciendo por primera vez:

/article/debugging-chrome/chrome-open-sources.svg: Command failed: Error reading SVG:Error domain 1 code 64 on line 2 column 6 of data: XML declaration allowed only at the start of the document identify-im6.q16: delegate failed `'rsvg-convert' -o '%o' '%i'' @ error/delegate.c/InvokeDelegate/1949. identify-im6.q16: unable to open file `/tmp/magick-128398AvZFRqNQ3lA': No such file or directory @ error/constitute.c/ReadImage/544.

El botón de activación (toggle button) abre la pestaña con los archivos.

y luego hello.js en el árbol de archivos.

Hagamos click allí y seleccionemos index.html

Estas son las zonas que se deberían ver:

La Zona de recursos lista los archivos HTML, JavaScript, CSS y otros, incluyendo imágenes que están incluidas en la página. Las extensiones de Chrome quizás también aparezcan aquí.

La Zona de Recursos muestra el código fuente de los archivos.

La Zona de información y control es para “debugging”, la exploraremos pronto.

**A Practical Guide to JavaScript Debugging**

ARNAB ROY CHOWDHURYMARCH 26, 2020 DEVELOPER TIPS, TRICKS & RESOURCES

Being a UI developer, I’ve learned one thing: It doesn’t matter how carefully you write your code. Suppose you’ve double-checked that you defined and called all functions the right way or followed all the best practices. Even then you’ll see that a tiny variable can sneak behind and create an error.

Now, suppose you find out that for some unknown reason a form validation or submit button isn’t working. Also, while going through the code, you’re unable to find the root cause of the error. Now, that’s really frustrating, isn’t it?

So what’s the solution to detect and fix the error? You guessed it right—debugging!

On that note, in this post, we’re going to discuss what debugging is and why we need it. We’ll also go over how to debug your JavaScript code using the browser’s developer console.

What Is Debugging?

No matter how much experience you have, the JavaScript code that you’re writing may contain an error. If not syntactical, there’s a high chance that you’ll get a logical error in case of a complex app.

A frustrating thing is that often you won’t get an error message or any clue about where the error is happening. Debugging is the technique of searching for and fixing these unknown errors.

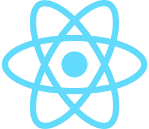
Debugging is done by forcing the code to stop running at a certain line. You can do that by setting a breakpoint. Once the code is paused, you can inspect and find out what’s wrong with the code. Wondering how to debug in the case of a JavaScript code? We’ll find out in the next section.

Fuente:<https://stackify.com/a-practical-guide-to-javascript-debugging/>

**TEMA 7**

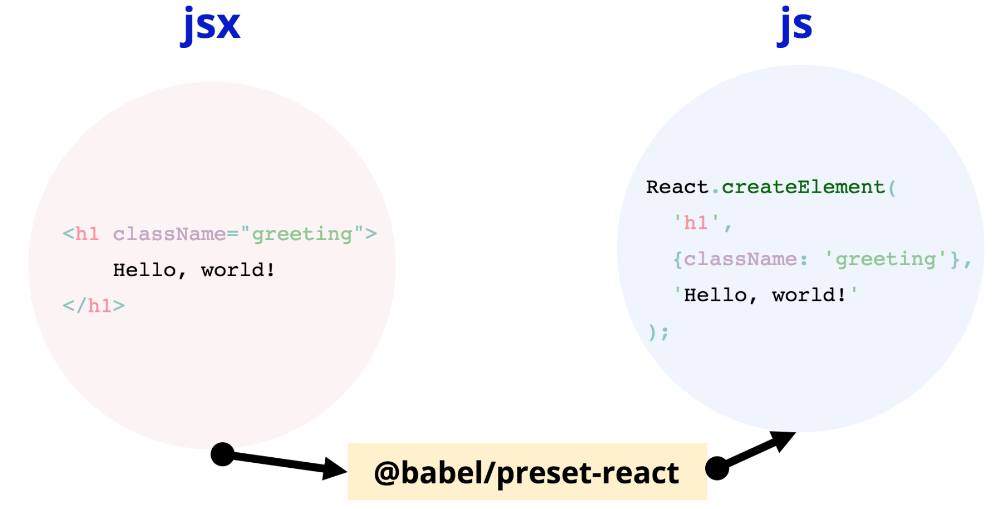
**Introducción a ReactJS**

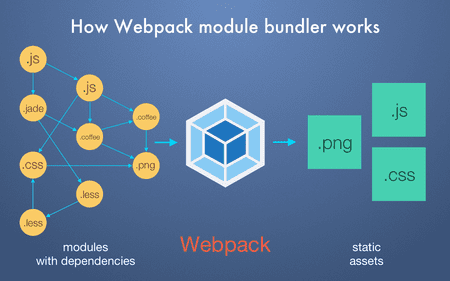




React no analiza las etiquetas y las convierte a HTML. Sin embargo, en realidad, JSX es solo una buena sintaxis para llamadas a funciones que crean elementos DOM.

**JSX es compilado ó transpilado a JavaScript**

 Los elementos JSX que se escriben son compilados en JavaScript mediante una herramienta llamada Babel. Babel es un compilador que transforma el código en JavaScript ES5 válido que todos los navegadores puedan entender, y está incluido en los proyectos creados con Create React App.



Después de ejecutar npm start, una herramienta llamada Webpack está buscando cambios en los archivos. Cuando lo hacen, alimenta esos archivos en Babel, que convierte JSX en JS, y lo envía al navegador a través del servidor de desarrollo que se ejecuta en el puerto 3000 por default.

Cada elemento JSX se convierte en una llamada de función, donde sus argumentos son sus atributos ("props") y su contenido ("children").

Aquí hay un ejemplo de un componente React simple que devuelve JSX:

function Hello() {

return <span> Hello! </span> ;

}

|  |  |
| --- | --- |
| Y aquí está el JavaScript generado por el compilador de Babel:  function Hello() {  return React.createElement(  'span',  {},  'Hello!'  );  } | La firma de la función React.createElement se ve así:  React.createElement(  string | element,  [propsObject],  [children...]  ) |

|  |  |  |
| --- | --- | --- |
| lenguaje de codificaciónEl **string|element** puede ser una cadena que describe una etiqueta HTML o SVG (como 'div' o 'span'), o puede ser una función componente (como HelloWorld, sin comillas). | El **propsObject y children** son opcionales, y también se puede proporcionar más de un hijo pasando argumentos adicionales:  function HelloWorld() {  return React.createElement(  'div',  {},  'Hello',  'World'  );  } | También se puede anidar las llamadas:  f function ManyChildren() {  return React.createElement('div', {},  React.createElement('div', {}, 'Child1'),  React.createElement('div', {}, 'Child2',  React.createElement('div', {}, 'Child2\_child')  )  );  }} |

Aquí hay algo un poco más complicado de JSX y una vista previa de que puede ser más complejo. Puede verse que hace referencia a un parámetro de función denominado **props**. Todavía no se habla de props, pero esta es la forma en que se pasa argumentos a los componentes React.

function SongName(props) {

return (

<span className='song-name' >

{props.song.name}

</span>

);

}}

**TEMA 8**

**Instalar CRA (create react app)**

Cómo configurar un proyecto de React con Create React App

**React** es un marco de trabajo de **JavaScript** popular para la creación de aplicaciones de **front-end**. Creado originalmente por Facebook, ganó popularidad al **permitir que los desarrolladores creen aplicaciones rápidas** usando **JSX**.

**ANTERIORMENTE**

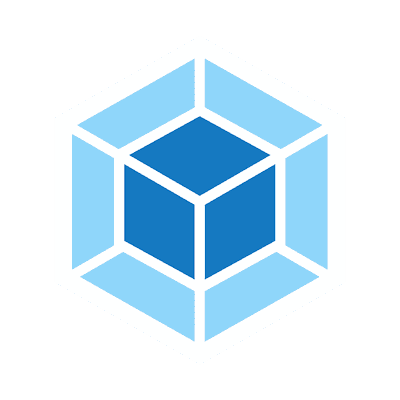
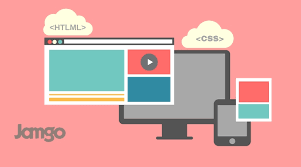
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| arma antigua icono gratis | Iniciar un nuevo proyecto de React solía ser un proceso complicado de varios pasos. | Implicaba la configuración de un sistema de compilación y un transpilador de código. | Esto para convertir la sintaxis moderna en código que todos los navegadores pudieran leer y una estructura de directorios básica. | computadora |

**ACTUALMENTE**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Create React App incluye todos los paquetes de JavaScript** que necesita para ejecutar un proyecto de React, como transpilación de código, linting básico, pruebas y sistemas de compilación. | También incluye un servidor con recarga directa que actualiza su página a medida que realiza cambios de código. | Por último, creará una estructura para sus directorios y componentes para poder comenzar a codificar en minutos. |

En otras palabras…

**X**



**X**

**X**



No tiene que preocuparse por la mayoría de los sistemas complejos de desarrollo de front-end moderno.

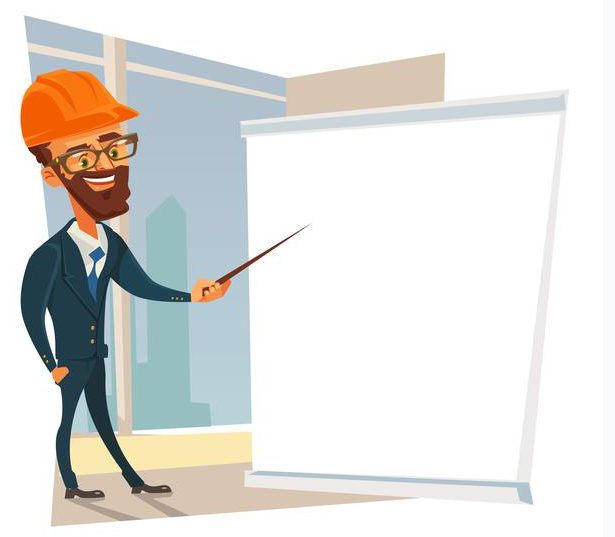
No necesita configurar Babel para que convierta a otro lenguaje su código de modo que se pueda utilizar en todos los navegadores.

No tiene que preocuparse por configurar un sistema de compilación como Webpack.



Puede comenzar a escribir código de React con una preparación mínima.

**Creemos un proyecto con Create React App**



Qué haremos:

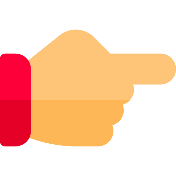
* Aplicación de React en ejecución que puede usar como base para cualquier aplicación futura.
* Los primeros cambios en el código de React.
* Actualizará los estilos y ejecutará una compilación para crear una versión totalmente reducida de su aplicación.

También usará un servidor con recarga directa para obtener comentarios al instante y analizará en profundidad las partes que componen un proyecto de React. Por último, comenzará a escribir componentes personalizados y a crear una estructura que pueda evolucionar y adaptarse a su proyecto.

**Necesitará lo siguiente:**

****

* **Node.js** versión 10.16.0 instalado en su computadora.
* También será útil tener **conocimientos básicos de JavaScript.**



Se creará una nueva aplicación usando el administrador de paquetes **npm** para ejecutar una secuencia de comandos remota. La secuencia de comandos copiará los archivos necesarios en un directorio nuevo e instalará todas las dependencias.

Esto significa que ejecutará el código de Create React App sin descargar el proyecto primero.

La aplicación npm instalará paquetes de JavaScript en su proyecto y también llevará un registro de los detalles de este.

Cuando instaló **Node**, también instaló una aplicación de administración de paquetes llamada **npm.**

La aplicación npm también incluye una herramienta denominada npx que ejecuta los paquetes ejecutables.

Una vez más, no es necesario que este directorio exista de antemano; el paquete ejecutable lo creará. La secuencia de comandos también ejecutará npm install dentro del directorio del proyecto, que descargará cualquier dependencia adicional.

El paquete ejecutable ejecutará la instalación de create-react-app en el directorio que especifique. Comenzará por crear un nuevo proyecto en un directorio que, en este tutorial, recibirá el nombre digital-ocean-tutorial.

Este comando iniciará un proceso de compilación que descargará el código básico junto con varias dependencias.

Para instalar el proyecto de base, ejecute el siguiente comando:

npx create-react-app digital-ocean-tutorial