

Del componente a la composición

"Cualquier persona puede escribir código entendible para una computadora. Un buen programador escribe código que los seres humanos puedan entender."

Martin Fowler – *Software developer and author.*

En la meeting pasada (así como en la toolbox anterior) aprendiste lo esencial sobre una pieza fundamental de construcción de React: el componente. Ahora es el momento de tomar el bloque y construir una casa.

React se promueve como una librería orientada al paradigma de componentes, es decir, que trata todo como un componente, desde un botón, hasta una vista entera. Además, propone un enfoque de construcción **declarativo**, lo cual significa que al implementar los componentes podemos indicar el qué son, en vez del cómo se construyen gracias al concepto de abstracción.

Toma de ejemplo la siguiente interfaz gráfica, y piensa cómo construirías este diseño con HTML.



Lo más probable es que utilices un `div` contenedor para una carta, y luego construyas los subcontenedores, por ejemplo:



```

<div class="card">
  <div class="header">
    <!-- Icono y botón de settings -->
  </div>
  <div class="tag">
    <!-- Etiqueta para "habilitado" -->
  </div>
  <div class="descripcion">
    <!-- Título y descripción -->
  </div>
</div>

```

Luego se repetiría este proceso tantas veces como la cantidad de cartas que existan en el diseño. Este enfoque es **imperativo**, ya que indica línea a línea cómo se debe construir la interfaz gráfica, con sus elementos HTML y respectivos estilos CSS.

El enfoque declarativo de React nos propone construir interfaces cuyo código se especifique en lo que es el componente, más no el cómo se construye. ¡Cuidado! Esto no significa que escaparemos de una estructura HTML declarativa, pero con React primero podremos hacer un molde, a través de una función, por ejemplo:

```

function Card() {
  return (
    <div>
      { /* El html aquí */ }
    </div>
  );
};

```

Para así poder implementar la carta de la siguiente forma:

```
  
<Card  
  icono={"connect"}  
  habilitado={true}  
  titulo="Demostración de la API"  
  descripcion="..."  
>
```

Esta implementación existirá tantas veces como cartas existan en el diseño. La gran diferencia de trabajar bajo este formato es que una vez que construimos el molde (la función) podremos crear componentes indicando lo que son (un `<Card />`) en vez de cómo se construye (toda la estructura HTML necesaria).

La habilidad de poder crear componentes gráficos con Javascript (y con JSX), nos da la ventaja de poder crear nuestras propias piezas de construcción, teniendo un control absoluto sobre cómo se comporta, cómo responde a interacciones, y especialmente, cómo se llama.

Detrás de la magia

Hasta ahora has podido realizar tus propios componentes utilizando React, pero, detrás de un buen truco de magia siempre hay una buena técnica.

Una aplicación web se caracteriza por ofrecer y manejar interacciones en su interfaz gráfica; si hacemos click sobre un botón, esperamos algo a cambio. Por ejemplo, una aplicación que sume dos números deberá ofrecer las interacciones necesarias para especificar los números, y sumarlos. Nuestro trabajo como *frontenders* consiste en programar desde la interfaz hasta los mecanismos necesarios para ofrecer tal funcionalidad.

Hasta ahora hemos visto cómo usar React para crear los componentes gráficos que puedan representar una sección de nuestra página, en las próximas meetings (y durante el resto del programa) nos dedicaremos a estudiar cómo construir los mecanismos de interacción para lograr ejemplos como el que se muestra a continuación:

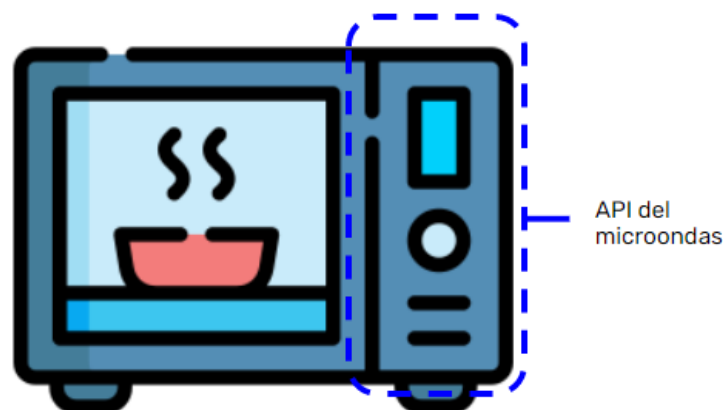




¿Cómo hace React para encontrar estos valores, y actualizar la interfaz en cuestión de segundos? Hay varias aristas a esta respuesta, pero queremos empezar contándote una: la [API 'document'](#). ¿API?

Las siglas API significan “Application Programming Interface”, lo cual se traduce a “Interfaz de programación de aplicaciones”. En este contexto, la “interfaz” no se refiere a una gráfica, sino más bien a la cara, o al frente de algo. El objetivo de una API es proveer abstracciones de métodos para ejecutar un procedimiento.

Para entender mejor el concepto de API, vamos a sacarlo fuera del entorno de desarrollo, y pensemos en un microondas. Sí, el que usas para calentar la comida.



El panel de opciones que presenta el microondas, que abarca desde los botones de tiempo, opciones para descongelar y métodos de cocción, entre otros, conforma la API para este objeto. Como personas usuarias del microondas, solo debemos presionar un botón y esperar el resultado. En desarrollo de software, las API's proveen funciones que, como desarrolladores, debemos invocar, y al igual que el microondas, esperar que esa abstracción ejecute algo.

Hay una [lista larga de API's](#), pero en este programa nos enfocaremos en un grupo pequeño. En esta toolbox te queremos contar de la **API 'document'**, la cual exploraremos a modo ilustrativo, para entender la magia de React.

Todos los navegadores vienen con la API 'document' la cual tiene un conjunto de métodos que nos permiten, a través de Javascript, buscar elementos HTML en



un documento y poder leer, insertar, borrar o modificar valores, o inclusive, mismos elementos. Dale un vistazo a este video para saber más sobre la API 'document':

[La API document](#)

¿Lo has visto? Bien, ahora te proponemos un mini challenge: [¿puedes mostrar por consola las dimensiones de tu ventana?](#) Sigue los pasos del video para realizar esto.

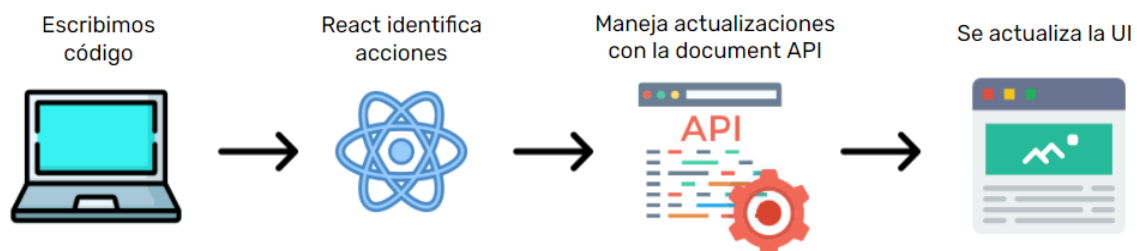
Como te mostramos en el video, la API 'document' provee una serie de métodos para manipular [el DOM](#) de forma dinámica con Javascript, lo cual significa que esta es una de las protagonistas en cualquier librería de frontend, ya que permite una comunicación directa con el documento HTML para obtener etiquetas, clases, o entidades a través de llamados de funciones. **En esencia, el funcionamiento de una aplicación web dinámica consiste en actualizar partes del DOM cuando ocurra una interacción.**

Es válido afirmar que podemos construir el frontend de una aplicación solo con esta API, pero ese trabajo sería lento, tedioso y poco escalable. Presta atención al siguiente ejemplo donde recreamos un ejercicio del Sprint 1, pero ahora con Javascript:

[Modificando el DOM con Javascript](#)

El virtual DOM

Cuando hacemos componentes, React actúa de intermediario entre la API 'document' y nosotros/as, los/as programadores. De esta forma podemos solo utilizar Javascript para construir los bloques, y de fondo, React se encargará de desplegar las instrucciones necesarias para actualizar la interfaz gráfica.



Sin embargo, este proceso no ocurre exactamente igual al ejemplo que te mostramos en el video anterior, es decir, React no atraviesa todo el DOM buscando elementos con clases o con id's para cambiar sus estilos y/o valores, sino que mantiene su propia representación del DOM en un objeto de Javascript llamado [Virtual DOM](#). Este objeto contiene un esquema exacto del documento HTML que conforma nuestra aplicación, incluyendo atributos y estilos.

La raíz de este objeto es un `div` con un id, el cual usualmente es `'root'` ("raíz" en español) y se ubica en el archivo `'index.html'`: `<div id="root">`. Dentro de este `div`, React monta todos los componentes que vayamos creando, y arma una estructura de árbol, similar al DOM tradicional que conocemos.

De esta forma React puede controlar con más facilidad todos los cambios necesarios que amerite nuestra aplicación, al buscar sólo dentro de un `div`, y no dentro del documento HTML. En el siguiente video te contamos cómo se conectan todas las piezas que hacen que nuestros componentes se vean en la interfaz gráfica.

[Estructura de una React App](#)

Nota: encontrarás el sandbox en la descripción del video.

Composición de componentes

La estructura general de una aplicación hecha con React (una [React App](#)) no difiere mucho de la estructura de un documento HTML. Ambos enfoques siguen el patrón de elementos dentro de elementos, creando este árbol invertido, donde un documento HTML tiene como raíz la etiqueta `<html>` mientras que una React app, tendrá el `<div id="root">` como nodo principal. Entender este esquema es esencial para desarrollar aplicaciones con React, y durante este sprint seguiremos explorando este concepto.

Si en las React apps, todo es un componente, podemos afirmar que construir interfaces gráficas a gran escala necesitan de una [composición de componentes](#).

Una composición no es más que la organización de varios componentes para construir un diseño. Hasta ahora hemos estudiado el componente como un elemento autocontenido, una card, como lo que te propusimos al final de la meeting pasado, el componente `<HotelInfo />`. El objetivo de este componente es reutilizarlo tantas veces como hoteles existan en nuestra base de datos.





Pero, un componente puede tener una función exclusiva, como por ejemplo, un header, footer o barra de búsqueda. Observa el siguiente mock:



En el mock tenemos 3 secciones principales que pueden trasladarse a componentes de una forma similar a esta:

```
<App>
  <Header />
  <BarraBusqueda />
  <Resultados />
</App>
```

Luego, dentro de cada componente, vamos a retornar el HTML necesario para mostrar la sección gráfica que corresponda. En el siguiente video te mostramos cómo podemos combinar los conceptos de CSS con los componentes de React.

[Layouts con componentes](#)

En resumen

La palabra “**abstracción**” atraviesa de forma transversal a las diferentes disciplinas que participan en el desarrollo de software. La abstracción le pone una cara, o interfaz, a un conjunto de funcionalidades haciéndolas accesibles y fáciles de entender.

En nuestra vida cotidiana, un control remoto nos ayuda a cambiar de canal con la sencilla acción de presentar un botón, y en nuestra vida como *frontenders*, invocar una función nos ayudará a construir un bloque de HTML de forma dinámica. En programación, estas abstracciones que proveen métodos para acceder a funcionalidades las llamamos API's. Sin embargo, en algunos casos, el concepto de abstracción va más allá de esto.


React se ayuda de la API 'document' para modificar el DOM de un sitio web, proveyendo a su vez una abstracción para los/as *frontenders* en forma de funciones de Javascript y JSX para poder interactuar con el DOM, creando componentes sin necesidad de escribir código que invoque a esta API.


El paradigma de componentes va más allá de crear estas piezas independientes y autocontenidas ya que podemos mezclar estos conceptos para construir *layouts* enteros y mezclar los conceptos que hemos aprendido sobre HTML y CSS para poder construir interfaces de aplicaciones dinámicas. Después de todo, React no es más que HTML, CSS y Javascript, con un montón de magia por detrás.




¡Prepárate para la próxima meeting!

Profundiza


 ¿Cómo “pensar en React?” Lee [este artículo](#) con pautas muy importantes que te ayudarán a adoptar una filosofía de componentización en aplicaciones web.

 Aprende más sobre el Virtual DOM en la [documentación oficial de React](#).

Herramientas

 Entender el DOM a veces es una tarea de imaginar, pero con esta herramienta, podemos copiar y pegar cualquier código HTML y analizar su estructura de árbol. Vista el [visualizador del DOM](#) para más información.

Challenge

 Construye [este layout](#) con React aplicando los conceptos de composición de componentes. En el mock, se han dejado propuestas sobre cómo puedes llamar a cada componente, así como un par de notas sobre los inputs que deberás de agregar.