

# ¡Llegó la hora de integrar!

*"El todo es más que la suma de sus partes"*

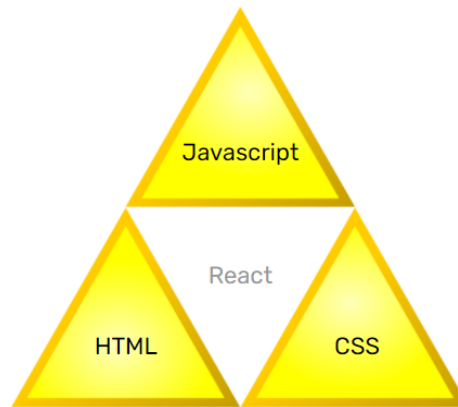
Aristóteles

## Hacia lo dinámico

No es nuestra intención sonar "dramáticos" pero esta toolbox marca un antes y un después en cómo venimos pensando las cosas en este programa. Durante todo este recorrido, hasta ahora, te hemos enseñado ciertas piezas del rompecabeza, y hoy tomamos unos pasos hacia atrás para ver la gran imagen. Y créenos, cuando la veas, te sorprenderas. Hoy es el día donde integramos todos los conocimientos en uno solo.



¿Recuerdas la triada tecnológica que te presentamos en la primera toolbox? Sí, la pirámide que representa los lenguajes que íbamos a aprender en este programa. Hoy vamos a colocar la pieza faltante en nuestro triángulo: [React](#).



Pero, antes de contarte qué es, y qué hace React, queremos tomarnos unas líneas para recordar brevemente lo que hemos aprendido, y qué problemas hemos podido solucionar con estos conocimientos.

Durante el primer sprint aprendiste todo sobre HTML y CSS, lo cual te permite construir páginas web simples e inmutables. Una vez “servidas”, es decir enviadas desde un servidor hacia nuestro dispositivo, éstas no cambian, a menos que realicemos una nueva petición.

También conociste las técnicas de diseño responsive, para adaptar las páginas a un dispositivo móvil. Este trabajo de construir sitios web inmutables y estáticos se le conoce como **maquetación**, y en sí solo, es un trabajo demandado en nuestra industria.



Luego, en el segundo sprint ya comenzaste a aprender sobre **programación**, de la mano de Javascript. Has puesto en práctica todos los tipos de valores permitidos por este lenguaje, haciendo énfasis en la programación funcional.





Hoy, es el momento donde combinamos la **maquetación** y la **programación** en un solo producto: **el frontend de una aplicación web**. Conectando Javascript con una interfaz gráfica podremos agregar interactividad, llevando nuestros sitios web a aplicaciones dinámicas con super poderes.

A partir de hoy empieza tu viaje hacia convertirte en unx frontend developer. A partir de hoy podrás crear aplicaciones web dinámicas e interactivas, y todo con la ayuda de, el aclamado, **React**.

## Mi sitio web... ¡reacciona!

Hay muchas definiciones de por que React tiene ese nombre, en Acámica nos gusta la idea de que este nombre se le ha dado debido a lo que le permite hacer a una aplicación: reaccionar ante cambios. Otras librerías de frontend también hacen lo mismo, como [Angular](#) por ejemplo, pero pensamos que React es un nombre *más cool*.



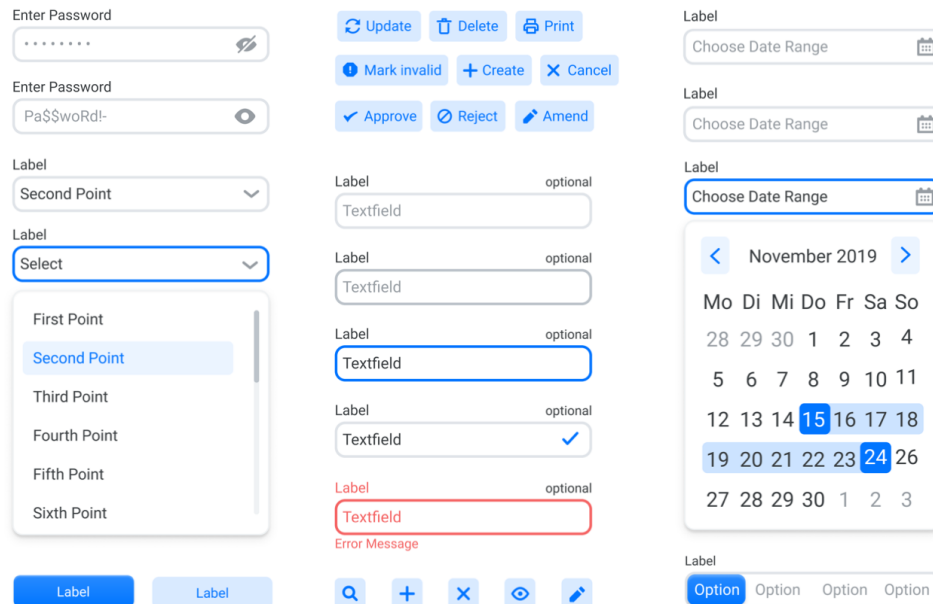
Una definición técnica de React es: “una librería de Javascript enfocada en construir interfaces gráficas, basada en el paradigma de componentes web”. Hay varios conceptos claves que necesitamos entender y practicar para sacar el máximo provecho que trae React. Estos conceptos son:

- Components
- JSX
- Props
- Eventos
- Estados

En la toolbox de hoy vamos a introducirnos en los primeros tres, mientras que el resto los iremos descubriendo en las próximas toolboxes y meetings. Empecemos con los **componentes**.

## Componentes

En React, un componente es la pieza fundamental de construcción de cualquier cosa que se encuentre en el frontend de una aplicación web. ¿Recuerdas cuando hablamos de los componentes gráficos en una interfaz?



Botones, date pickers, checkboxes, radio inputs, títulos, todos son componentes. Una interfaz gráfica se compone de un conjunto de componentes como los anteriores, y en React, todo, **absolutamente todo**, es un componente, inclusive, el contenedor donde existe toda la aplicación.



Pero, a diferencia de HTML y sus etiquetas como `<button>` o `<input>`, en React podremos crear **nuestros propios componentes**. Estos estarán contruidos en HTML y CSS permitiendo ofrecer interacciones con Javascript. La mejor parte de todo esto es que lo que necesitas para codear en React, ¡ya lo sabes!

¿Aún no te lo imaginas? Lo siguiente te sorprenderá: en React, los componentes se crean a través de funciones de Javascript, y estas funciones tendrán la habilidad de retornar el HTML necesario para crear ese componente propio. De seguro tienes muchas preguntas, y antes de avanzar, queremos mostrarte este video donde evidenciamos lo anterior.

[Componentes en React](#)

## JSX

Después de los componentes, el **JSX** es el segundo concepto del cual te queremos hablar en esta toolbox. Las siglas JSX significan “Extended Javascript”, y es la forma que tiene React de permitir a los/as frontenders escribir código HTML dentro de una función.

Como te mencionamos en el video, cuando escribimos JSX, debemos considerar retornar todo dentro de un nodo HTML, siendo este, cualquier etiqueta estructural, como un `<div>`. Considera los siguientes ejemplos para entender un poco más este concepto.

Supongamos que estamos creando un componente que muestra en pantalla tres botones, la forma correcta sería:

```
function Botones() {  
  return (  
    <div>  
      <button>A</button>  
      <button>B</button>  
      <button>C</button>  
    </div>  
  )  
}
```



Si nos olvidamos de contener el HTML, en un nodo raíz, **React nos gritará**. Intentemos no hacer enojar a React **tanto**.

```
function Botones() {  
  return (  
    <button>A</button>  
    <button>B</button>  
    <button>C</button>  
  )  
}
```

Sin embargo, esto no es un capricho, esto se necesita para garantizar un buen performance a la hora de actualizar la interfaz gráfica. Más adelante hablaremos de esto, pero si quieres saber más, te dejamos [este artículo de Medium](#).

Como alternativa a las etiquetas raíz que todo componente debe tener, React ofrece los **Fragmentos**, etiquetas especiales que no tienen ningún tipo de estilos o peso dentro del HTML. El fragmento puede ser escrito a través de la etiqueta especial `<React.Fragment>` o sencillamente, una etiqueta vacía como `<>`. Presta atención a los siguientes ejemplos, ambos son válidos:

```
function Botones() {  
  return (  
    <>  
      <button>A</button>  
      <button>B</button>  
      <button>C</button>  
    </>  
  )  
}
```

```
function Botones() {  
  return (  
    <React.Fragment>  
      <button>A</button>  
      <button>B</button>  
      <button>C</button>  
    </React.Fragment>  
  )  
}
```

La sintaxis de JSX permite más que poder escribir HTML dentro de una función de Javascript, ya que también permite escribir el mismo lenguaje dentro del HTML. En el siguiente video te contamos más sobre esto.

[JSX en React](#)



## Composiciones

Los ejemplos previos están contenidos dentro de una función, pero sabemos que en una interfaz gráfica, y más aún en una aplicación web, hay más de un componente. React, de la mano de Javascript, ofrece un modelo de composición poderoso que permite la reutilización de sus componentes dentro de otros componentes. En español, esto no es más que funciones ejecutadas dentro de otras funciones.

En Javascript, cuando creamos funciones y las queremos ejecutar, las **instanciamos**, para procesar las instrucciones que fueron definidas dentro de su cuerpo.

```
// definición
function suma(x,y) {
  return x + y
}

// instanciación
suma(2,2) // 4
```

Te mencionamos que los componentes de React son solo funciones de Javascript, por lo tanto, el concepto anterior aplica también, pero con una diferencia, la cual consiste en que la instanciación de los componentes se escribe como una etiqueta HTML.

```
// definición
function Suma() {
  let suma = 2 + 2;
  return <p>La suma de 2 + 2 es {suma}</p>
}

// instanciación
<Suma />
```

## Props

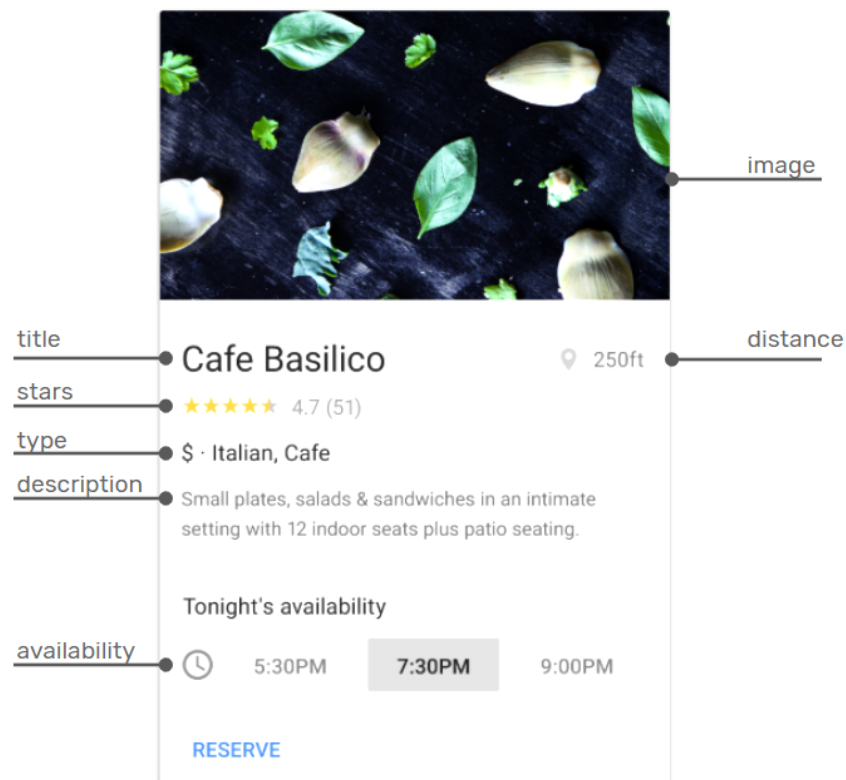
El último concepto sobre el que queremos comentarte es el de las **props**, o propiedades de un componente. ¿Recuerdas los argumentos con los que puede trabajar una función? Pues es el mismo concepto para los componentes de React.

Las props son parámetros que pueden ser especificados tanto en la instanciación de un componente, como en la definición del mismo. Las **props** son almacenadas en un objeto, el cual está disponible para todos los componentes creados, sean especificadas o no. En el caso de que no se especifique ninguna prop para un componente, este objeto estará vacío.

En el siguiente video te mostramos cómo trabajar con las props dentro de un componente de React, así como también los conceptos de composición de componentes.

[Props](#)

Llevando este concepto de props a un nivel más gráfico, piensa en un componente `<Card />`, el cual puede lucir así:







Supongamos que tenemos una aplicación que muestra una lista de cartas, cada una con información de un restaurante o café. Podemos usar la misma estructura de una prop, con distintos valores, para reutilizar el `<Card />` component y mostrar información distinta en cada instanciación. La instanciación puede verse de la siguiente manera:

```
<Card
  title={"Cafe Basilico"}
  stars={4.5}
  type={"italian"}
  description={"Small plates, salads..."}
/>
```

¿Puedes identificar cómo se relacionan las props en el código, y en el componente renderizado? Cada prop especificada en el código se representa en la UI.

## ¡Prepárate para la próxima meeting!


### Profundiza

-  [Componentes y props por la documentación oficial de React.](#)
-  [Artículo sobre cómo “pensar” en React](#) (¡recomendado!).

### Comunidad


-  Visita el [subreddit de React](#) e involúcrate en la discusión.

### Challenge

 ¡Componetiza una interfaz! Queremos que revises [esta interfaz gráfica](#) y nos des tu opinión, según lo aprendido hasta ahora, sobre qué componentes gráficos identificas. También, lista las props de cada uno de estos componentes, junto a sus valores.



Mantén tu bosquejo cerca ya que estaremos visitando este ejemplo de nuevo para identificar otros conceptos.

 ¿Quieres empezar ya con React? Hemos preparado [este sandbox](#) con un componente andando. Sigue las instrucciones y completa las consignas. Opcionalmente tendrás un challenge bonus. ¿Te animas?

