

DESARROLLO DE UNA LIBRERÍA NPM  
DE FRONT-END BASADA EN  
COMPONENTES DE REACT PARA EL  
ÁGIL MAQUETADO DE UNA PÁGINA  
WEB USANDO BUENAS PRÁCTICAS DE  
DISEÑO UX/UI

Fernando Garcia Corona

<https://github.com/FerCorona/crown>

22 de noviembre de 2020

# Índice general

<b>1. INTRODUCCIÓN</b>	<b>3</b>
<b>2. JUSTIFICACIÓN</b>	<b>4</b>
<b>3. OBJETIVOS</b>	<b>10</b>
3.1. OBJETIVOS GENERALES . . . . .	10
3.2. OBJETIVOS ESPECÍFICOS . . . . .	10
<b>4. ALCANCES Y LIMITACIONES</b>	<b>12</b>
<b>5. MARCO TEÓRICO</b>	<b>13</b>
<b>6. PROCEDIMIENTO Y DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS</b>	<b>18</b>
6.1. CREACIÓN DEL AMBIENTE DE DESARROLLO . . . . .	21
6.1.1. AMBIENTE DE DESARROLLO . . . . .	21
6.1.2. INICIALIZACIÓN DEL ARCHIVO NPM . . . . .	22
6.1.3. INICIALIZACIÓN DE GIT EN NUESTRO PROYECTO . . . . .	26
6.1.4. CONFIGURACIÓN WEBPACK . . . . .	27

6.1.5.	CONFIGURACIÓN BABEL . . . . .	31
6.1.6.	AGREGAR REACT AL PROYECTO . . . . .	32
6.1.7.	CONFIGURACIÓN ESLINT . . . . .	33
6.2.	DESARROLLO . . . . .	36
6.2.1.	CONFIGURACIÓN ARCHIVO INICIAL . . . . .	36
6.2.2.	ELEMENTO BOTÓN . . . . .	37
6.2.3.	ELEMENTO LABEL . . . . .	41
6.2.4.	ELEMENTO INPUT TEX . . . . .	43
6.2.5.	ELEMENTO DROP DOWN . . . . .	47
6.2.6.	ELEMENTO RADIO BUTTON . . . . .	47
6.2.7.	ELEMENTO SWITCH . . . . .	47
6.2.8.	ELEMENTO TABLE . . . . .	47
6.2.9.	ELEMENTO CHECK BOX . . . . .	47
6.2.10.	ELEMENTO IMAGE . . . . .	47
6.3.	TEST . . . . .	47
<b>7.</b>	<b>RESULTADOS</b>	<b>48</b>
<b>8.</b>	<b>CONCLUSIONES Y RECOMENDACIONES</b>	<b>49</b>
<b>9.</b>	<b>FUENTES DE INFORMACIÓN</b>	<b>50</b>
<b>10.</b>	<b>ANEXOS</b>	<b>51</b>

# INTRODUCCIÓN

# JUSTIFICACIÓN

Actualmente, el desarrollo de páginas web es una de las oportunidades de empleo con mayor demanda dentro del área de la informática, en los principales buscadores de empleo se pueden encontrar gran cantidad para ejercer dentro esta rama. El desarrollo Front-end, de páginas web, es una de las subáreas más buscadas por las empresas que implementan servicios basados en web, o que de alguna manera consumen algún producto. El famoso sitio Stackoverflow realiza una serie de estadísticas anuales en las cuales da a conocer los lenguajes de programación en los cuales ellos obtienen un mayor número de búsquedas y respuestas. Las estadísticas muestran que, durante el transcurso del actual año (2020), el lenguaje más usado por la comunidad de desarrolladores profesionales es JAVASCRIPT. Por otra parte, dentro esta serie de estadísticas se cuenta con un apartado para los Frameworks, en el cual encontramos que en segundo puesto está REACT, solo por debajo de JQUERY el cual puede ser embebido dentro de REACT. Finalmente se encontró que la librería número uno es Node.js. El desarrollo Front-end de páginas web consiste en hacer la visualización que tenemos cuando ingresamos a algún sitio desde nuestro navegador, para esto es necesario que

el programador que realiza la tarea tenga conocimientos básicos de HTML, CSS y JAVASCRIPT, para que sea posible construir una web sencilla. Cabe destacar que un desarrollador Front-end no es el encargado de diseñar la experiencia de usuario ni tampoco el diseño de interfaz gráfica, ya que para esto existen otras disciplinas especializadas, pero en caso de tener conocimiento en el área puede agregar una herramienta que puede combinarse y agregar habilidades.

En la medida que una web escala, esta tiende a aumentar su complejidad de desarrollo si no se comienzan a usar librerías o frameworks, que nos permiten a tener un trabajo más limpio, organizado, seguro y modulable. Como se mencionó anteriormente JAVASCRIPT es usado tanto de manera profesional como con otros fines como los académicos, este lenguaje cuenta con un gestor de paquetes denominado NPM, que permite que se agreguen miles de funcionalidades extras a tu proyecto, NPM consiste en un cliente de líneas de comandos con el cual es posible agregar a nuestro proyecto paquetes, estos paquetes son de utilidad porque podemos reusar código que alguien más ya desarrolló, probó y decidió compartirlo, haciendo que el trabajo sea más ágil. De igual manera nosotros podemos aportar publicando nuestra librería. Dentro de NPM contamos con miles de librerías de código abierto. Teniendo esto en cuenta, nos da la posibilidad de tener nuestra propia librería y publicarla, para que personas a las cuales tienen alguna necesidad, pero no cuentan con el tiempo de ejecutarla puedan acceder a la nuestra, e incluso nosotros mismos usarlas en posteriores proyectos.

El desarrollo de software de código abierto consiste en publicar algún tipo de herramienta propia, el cual será de licencia pública para que más personas

puedan acceder al código fuente, si lo desean podrán usarlo o adecuarlo a sus necesidades. Anteriormente se mencionó un Framework de Front-end llamado REACT, este es mantenido por Facebook desde mayo del 2013 que fue su fecha de publicación, en la actualidad tiene más de mil contribuidores según lo indica el repositorio oficial. Algunas de las particularidades de REACT es que nos motiva a crear componentes que pueden ser utilizados más veces, y de esta manera tener una menor cantidad de código y más reutilizable. Nos deja crear una aplicación en una sola página que de ser de una manera tradicional y a gran escala se convertirían en una tarea imposible.

Por el uso desmedido de cada una de las tecnologías que se han mencionado nace la razón por la cual se desea desarrollar una librería de NPM, la cual ayudará grandemente a la comunidad de desarrolladores de páginas web y esto principalmente a las personas que cubren el rol de programadores Front-end. Esta librería permitiría a los desarrolladores agilizar su carga de trabajo, poniendo a su alcance un conjunto de elementos usados en el desarrollo Front-end, alguno de ellos son botones, textos, etiquetas de texto, tablas, checkbox, radio botones, etc. Los cuáles serán elementos definidos, que contarán con una definición de estilos (SCSS) establecidos, cada componente permitirá al desarrollador modificar parámetros básicos como el color, el texto y acción que va a realizar, esto con el fin de adaptarlo a las necesidades propias del proyecto en el que se va a hacer la implementación de la librería. Otra ventaja por parte de la librería es que esta estará basada en prácticas modernas del diseño UI/UX, como lo es Mobile First Indexing (ideología de Google), que nos pide enfocar el diseño de cualquier web primero para dispositivos móviles, ya que afirman que es en el mercado el cual consume más contenido

web, ayudándonos a no requerir un conocimiento avanzado sobre el diseño de interfaces gráficas. Finalmente, además de los elementos básicos que incluirá la librería, tendrá más elementos que permitirán aún más la eficiencia, contendrá otros elementos compuestos como formularios, tarjetas, alertas, pies de página, menús de navegación, elementos deslizables.

La última parte de la librería consistirá en elementos aún más compuestos, denominados plantillas que consiste en pantallas de login, registro, página de inicio entre otras. Abriendo la posibilidad que más personas puedan aportar creando sus propias plantillas, y crear una comunidad de desarrollo. Debemos tomar en cuenta que este proyecto no se puede clasificar dentro del área de los frameworks ya que para ser parte de, es necesario contemplar toda la estructura necesaria dentro de una página web, modelos, vistas y controladores (MVC). Nuestro proyecto está focalizado en las vistas, por lo cual puede clasificarse como librería.

Con el uso de esta librería reduciremos el tiempo de desarrollo, ya que no comenzaremos a escribir HTML y CSS desde cero, tendremos una base común sobre la cual podemos seguir. Todo el equipo tendrá los mismos estilos y evitaremos que nuestra web tenga discrepancias dentro de los diferentes módulos o vistas que tiene nuestra web. Al usar esta librería nos aseguraremos de que las vistas de nuestra aplicación web puedan mirarse estéticamente iguales y tendremos la seguridad de que luzca de la misma forma en Chrome, Safari Firefox o un mayor número de navegadores, no importa si es una versión actual o una más antigua. Con esta librería evitamos el tiempo de aprender un nuevo Framework, ya que funciona sobre REACT y los conocimientos necesarios son saber REACT. Tendríamos una organización predeterminada



para todo el proyecto en el que se implementa, teniendo uniformidad en el desarrollo y en la interfaz gráfica, aumentando la agilidad en el trabajo y de manera proporcional reduciendo el tiempo y primordialmente el costo que implica, también reducirá el mantenimiento, así como los posibles errores. Debemos tomar en cuenta que el número de personas que participan en el desarrollo de un software es variado, pero regularmente este es mayor a uno, por lo que es conveniente que el código sea claro, legible y reutilizable para que el mayor número de personas que están involucradas, para que puedan trabajar en la mayor parte del proyecto sin problemas en caso de que exista un cambio de roles, es por lo que usar una librería estandariza el trabajo. Las librerías cumplen con gran cantidad de pruebas, para que no surjan problemas y sean impedimento al implementarlas, esta no será la excepción al incluir pruebas de validación, así también se eliminan errores en el proyecto donde se implementan.

Lamentablemente como en todo framework y librería existe la desventaja a que al implementarlo este te fuerza a verte limitado con respecto a la cantidad de configuración y personalización que podemos editar. Otra cosa por tomar en cuenta es que no se necesita aprender una sintaxis específica para usar esta librería, pero es necesario conocer la sintaxis que se usa con REACT, ya que esta librería es basada en REACT, y es necesario conocer los conceptos básicos como el uso de componentes y los props. Al usar una nueva librería es posible que sientas que estás perdiendo el tiempo al incorporarlo en proyectos, debido al tiempo de aprendizaje. También no se encuentra recomendable agregar la librería en un proyecto que ya está avanzado, y no lo es por el funcionamiento o incompatibilidad de librerías, esto es

más bien, porque no es viable visualmente ya que se encontrara elementos y vistas distintas unas entre otras, esto, si no se decide actualizar los elementos existentes lo cual aumenta el tiempo de desarrollo.

Debemos considerar que al agregar una capa software extra a nuestro Proyecto aumenta en tamaño, ya que además del peso de REACT que es necesaria para usar esta librería debemos contar el peso de nuestra librería en sí, aunque no todas las tecnologías mencionadas son agregadas en la versión que estará alojado en el proyecto, ya que por ejemplo WEBPACK, solo es usado durante el desarrollo.

# OBJETIVOS

## 3.1. OBJETIVOS GENERALES

Desarrollar una librería para el desarrollo de páginas web, especialmente para el área de front-end, publicada en el gestor de paquetes de JAVASCRIPT “npm”, usando el framework REACT, permitiéndonos la utilización de componentes para facilitar, agilizar y mejorar el maquetado de una página web, incluyendo prácticas modernas de UX/UI para el diseño de interfaces y experiencia de usuario.

## 3.2. OBJETIVOS ESPECÍFICOS

Desarrollar una librería para el desarrollo front-end de páginas web, para el gestor de paquetes de JAVASCRIPT llamado npm, usando el framework REACT que nos permite la creación de componentes reutilizables. Para el correcto funcionamiento de la librería se implementará web-pack que nos permite empaquetar y exportar todos los módulos y dependencia que incluye nuestra librería en un solo archivo para la correcta y ágil implementación.

Aunado a esto se utilizará babel que es un convertidor de código JAVASCRIPT a versiones anteriores, lo que nos permitirá una gran cantidad de compatibilidad con navegadores antiguos. Para unificar nuestra sintaxis se usará ESLINT el cual nos permite definir una guía de estilos para la librería, sobre esto se usará una guía de estilos ya definida y probada, la de AIRBNB.

# **ALCANCES Y LIMITACIONES**

# MARCO TEÓRICO

- **Librería Front-end:** Una librería Front-end es una herramienta que es agregada a nuestros proyectos, en este caso web, el cual incorpora elementos que otras personas o equipos ya desarrollaron, sumando funcionalidad a nuestra web, reduce el mantenimiento y el tiempo de desarrollo, algunas características que existen actualmente y pueden agregarse van desde gráficas, animaciones, mapas etc.
- **NPM :** NPM es un gestor de paquetes que nos permite agregar dependencias a cualquier proyecto basado en JAVASCRIPT, esto es posible con un cliente de líneas de comandos que es útil para poner o quitar los paquetes que deseamos. La configuración consta de un archivo en el cual contiene una lista de las dependencias que se quieren instalar en nuestro proyecto. Actualmente existen miles de paquetes que pueden ser descargados de manera gratuita y de igual manera permite colaborar.
- **REACT :** REACT es una librería de JAVASCRIPT para el desarrollo de interfaces de usuario (Front-end), famosa por la posibilidad de

hacer fácilmente webs de una sola página. REACT fue desarrollada por Facebook y con el paso del tiempo, conocidas empresas han empezado a implementarlo en sus proyectos. REACT nos permite desarrollar de una forma más ordenada y con el menos código necesario. Se considera que REACT no es un Framework en comparación con ANGULAR o EMBER porque no tiene cada una de las áreas que propone el modelo vista controlador, este solo se encarga de las vistas. Uno de los puntos fuertes de REACT es que cuenta con un DOM virtual el cual es almacenado en memoria, esto provoca que cuando algo cambie este se vea reflejado en memoria de una forma más rápida, después el DOM virtual será comparado con el DOM del navegador y solo actualizará lo que encuentre diferente. REACT funciona en base a componentes que pueden ser reusados cuantas veces sea necesario, una forma fácil de entenderlo puede ser como la programación orientada a objetos al hacer una instancia de una clase, estos componentes pueden tener un estado, que es encapsulado por sí solo, de manera local por cada uno y puede ser actualizado.

- **HTML** : HTML es un lenguaje con un conjunto de etiquetas que permite definir el contenido con el que podemos interactuar dentro de una página web.
- **SASS** : SASS es un preprocesador de CSS, el cual nos ayuda a agregar características que no tiene CSS puro y que son propias de los lenguajes de programación como variables, funciones, herencia entre otros. Nos permite dedicar menos tiempo para mantener y crear el CSS y agrega

la posibilidad de tener una organización modular.

- **WEBPACK** : Hace algunos años JAVASCRIPT iniciaba como un lenguaje que nos permitía agregar interacción a nuestras páginas web, que anteriormente eran simplemente contenido estático. JAVASCRIPT nos permitía recuperar los datos que eran introducidos en formularios, podíamos mostrar ventanas emergentes o incluso agregar animaciones. Para agregar JAVASCRIPT en nuestros archivos de HTML, se debía agregar la etiqueta `<script>`, donde se indicaba la ruta donde estaba almacenado nuestro archivo .js de JAVASCRIPT. Años más tarde se agregó soporte al lenguaje JAVASCRIPT, para permitir hacer peticiones asíncronas a nuestros servidores, esto hizo que las empresas empezaran a ver como viable el desarrollo web, que previamente estaba enfocado en el desarrollo de escritorio, también hizo que los archivos .js empezaran a crecer en cantidad de líneas en nuestros proyectos, y se miraba reflejado en nuestros archivos .html los cuales empezaron a tener múltiples etiquetas `<script>`. Lo que significaba múltiples peticiones para obtener los archivos del servidor, esto agregaba una capa de complejidad, debido a la inclusión de múltiples archivos .js embebidos en un .html y al agregarlos se debe tener en cuenta el orden en que se listan, porque es posible que tengan dependencias entre ellos. Actualmente es posible comprimir los múltiples archivos .js para que sean menos peticiones al servidor (CDN). Cuando un proyecto con JAVASCRIPT crecía aumentaba el número de módulos que se agregan, pero no existía la forma de gestionarlos. Un problema que tenía JAVASCRIPT es que



los navegadores no soportan un sistema nativo de módulos por eso se agregaron Require.js y Common JS. En base a esto nació WEBPACK que permite usar NPM como gestor de dependencias y soporta el sistema de módulos Common JS. WEBPACK permite modular nuestro código, para esto genera un grafo dado un punto de entrada, el punto de entrada es el nodo inicial de nuestro grafo, el grafo es generado con las inclusiones de elementos encontrados en los archivos, como son imágenes, archivos de CSS etc. Esto genera un archivo final en el que estará el empaquetado de nuestros archivos. WEBPACK nos permite especificar cargadores para distintos tipos de archivos como imágenes, SCSS, CSS, HTML, JSX, etc. ya que nativamente solo admite archivos JAVASCRIPT. La configuración es dada por un archivo que se coloca en la raíz del proyecto.

- **BABEL** : BABEL es un traductor de código JAVASCRIPT, que permite convertir el código JAVASCRIPT más moderno en alguna versión más vieja, esto nos proporcionó la capacidad de que nuestro código pueda ser ejecutado por más cantidad de navegadores, los cuales solo pueden interpretar versiones anteriores de JAVASCRIPT. Por ejemplo, el siguiente código es usado en REACT para mostrar un texto.

A code editor window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. The code inside is: 

```
return (  
  <div>  
    ¡Soy Pony!  
  </div>  
);
```

El código será convertido en el que se muestra a continuación, el cual es interpretada por un mayor número de navegadores.

A code editor window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. The code inside is: 

```
return REACT.DOM.div(null, '¡Soy Pony!');
```

- **ESLINT** :ESLINT nos permite definir una guía de estilos en nuestro código, lo que nos ayudará a tener un código limpio y claro para que sea fácil de editar y mantener, podemos agregar guías de estilos una de ellas es la de AIRBNB, que es una de las más usadas.

# PROCEDIMIENTO Y DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS

El objetivo de desarrollo de la presente librería se basa en el siguiente principio, “ cuando un elemento sea actualizado, sea un botón, cuadro de estado, entrada de texto, etc. el componente regresará un objeto el cual tendrá el nombre del elemento y el nuevo valor para que este sea actualizado en el state de react ”. Con esto deseamos generar el siguiente flujo.

```
import React, { Component } from 'react';
import { CheckBox } from 'crown';

class App extends Component {

  constructor(props) {
    super();
    this.state = {
      aceptar: ''
    }
  }

  render() {
    return (
      <CheckBox
        text='Enviar datos a mi correo'
        onChange={this.updateState}
        isChecked={this.state.aceptar}
        stateName={'aceptar'}/>
    );
  }
}

export default App;
```

Importamos el elemento `CheckBox` desde “crown” dentro de el método `render` en `return` ponemos el componente `CheckBox` y le damos los siguientes props.

- **onChange:** Función que actualiza el estado, que definiremos adelante.
- **isChecked::** Ponemos el valor de “aceptar” que está en el estado.
- **stateName:** Es el nombre ( y no el valor como en `isChecked` ) con el que identificamos el elemento en el estado.

Dentro del componente `CheckBox` se devuelve un objeto como el que se muestra enseguida.



```
onChange={() => llamadaDeRegreso({  
  stateName: 'Boton1',  
  value: false  
})}
```

- **onChange:** Este es el evento que se ejecuta cuando una acción es disparada, por ejemplo cuando haces click en un CheckBox.
- **llamadaDeRegreso:** Es la función que recibe el componente, y cuando se hace click en un CheckBox este, por defecto llama la acción que está dentro del evento onChange. En este caso llama a llamadaDeRegreso.
- **stateName:** Es el nombre ( y no el valor como en isChecked ) con el que identificamos el elemento en el estado.
- **value:** Este dato almacena el valor actualizado del elemento, esto define si el CheckBox está seleccionado o no.

Esto facilitará a tener una única función capaz de actualizar todos los elemento que agregamos ya que contamos con el nombre, que es con el que se identifica en el estado y también el nuevo valor.

A code editor window with a white background and a light gray border. It features three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a monospaced font with syntax highlighting: 

```
updateState(event) {  
  this.setState({  
    [event.stateName]: event.value  
  });  
}
```

## 6.1. CREACIÓN DEL AMBIENTE DE DESARROLLO

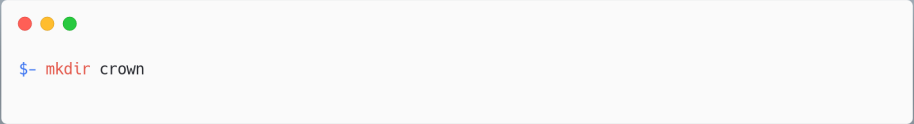
### 6.1.1. AMBIENTE DE DESARROLLO

Durante el desarrollo de la presente librería se usó el sistema operativo MacOS, con la terminal que se incorpora por defecto en el mismo. El primer paso que se debe realizarse es colocarse en el directorio en el que se desea trabajar, esta librería se sitúa en la carpeta Documentos durante el desarrollo, usando el siguiente comando es posible cambiar de directorio, ejecutandolo en la terminal.

A terminal window with a white background and a light gray border. It features three colored window control buttons (red, yellow, green) in the top-left corner. The prompt is a blue dollar sign followed by a hyphen. The command is `cd Documentos` in red text.

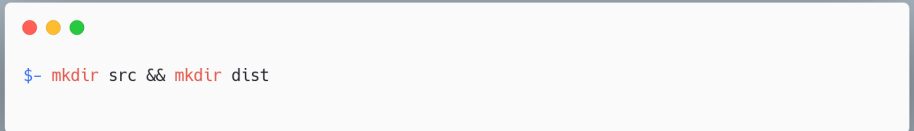
```
$- cd Documentos
```

Después se creó la carpeta de desarrollo con el siguiente comando. Se llamó de manera simbólica como “crown”, que significa en español “corona”.

A terminal window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. The command prompt is '\$-' followed by the command 'mkdir crown' in a monospaced font.

```
$- mkdir crown
```

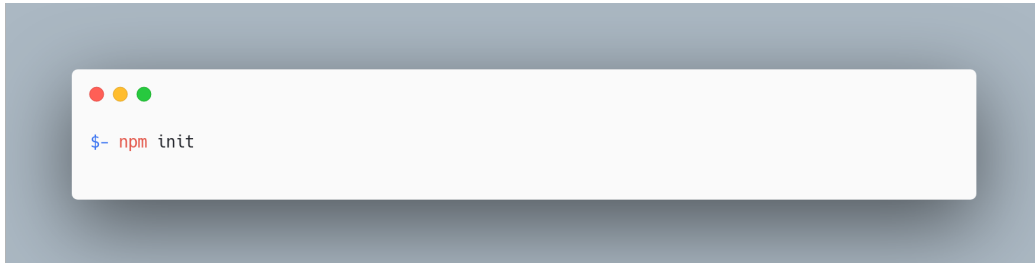
Dentro de esta carpeta debemos crear dos carpetas, una tendrá el código fuente, y la otra tendrá el resultado del código procesado que se importará por otros proyectos. Solo debemos ejecutar el siguiente comando dentro de “crown”.

A terminal window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. The command prompt is '\$-' followed by the command 'mkdir src && mkdir dist' in a monospaced font.

```
$- mkdir src && mkdir dist
```

### 6.1.2. INICIALIZACIÓN DEL ARCHIVO NPM

Se continuó inicializando el archivo de NPM, el cual nos sirve para llevar el control de las dependencias de JAVASCRIPT que se vayan agregando, esto ejecutando el siguiente comando.



Al introducir el comando anterior este preguntara por una lista de datos necesarios, para tener el control de los paquetes, estos son los siguientes datos que se deben proporcionar:

- **Nombre del paquete:** Este es el nombre simbólico con el cual se puede identificar el paquete dentro del buscador de NPM, por lo tanto, es el nombre con el que nuestro paquete será encontrado.
- **Versión del paquete:** Con esta opción controlaremos la versión, y en caso de que se agreguen funcionalidades o se resuelva algún error, tendremos una manera de actualizar en los proyectos que incorporen esta librería.
- **Descripción del paquete:** Daremos a las personas una muy breve explicación acerca del uso que puedes obtener con nuestra librería.
- **Punto de entrada del paquete:** Es el directorio el cual será importado cuando agreguemos nuestra librería a otros proyectos, este podrá incluir la lógica o que solamente sea el nodo inicial de todo nuestro código.
- **Comando de prueba:** Dentro de este archivo nos permite incluir



comandos que afectan a nuestra librería, en este caso, este comando nos sirve para ejecutar una serie de pruebas, para usar antes de publicar una nueva versión.

- **Repositorio de GIT del paquete:** Dentro de esta línea, debemos poner la dirección url en el cual está alojado nuestro proyecto. Este será agregado más adelante junto con el archivo de configuración de GIT.
- **Palabras clave del paquete:** Es una lista de palabras la cual nos ayuda para el momento cuando se inserta una búsqueda en el gestor de NPM, y pueda realizar una búsqueda basada en las palabras que describen la utilidad de nuestro paquete.
- **Autor del paquete:** Es el nombre del autor, autores u organización la cual está desarrollando el proyecto.
- **Licencia del paquete:** Existen una serie de licencias posibles a ser seleccionadas, para este caso se eligió la licencia MIT (MIT, Massachusetts Institute of Technology), que es una licencia de software que fue originada por el Instituto Tecnológico de Massachusetts, significa que el código que es producido bajo esta licencia es de uso libre, con la que damos muy pocas limitaciones de reutilización del código

En la siguiente imagen se muestra un ejemplo de los datos solicitados por el comando y los datos introducidos, los cuales son de prueba y nos son los mismos que se ingresaron el proyecto original. Todo esto generará un archivo final llamado package.json en el directorio raíz, este contendrá la configura-

ción dada en este paso. Finalmente preguntará si la información introducida es correcta y nos confirmara con una impresión en consola de los datos que estarán almacenados en el archivo.

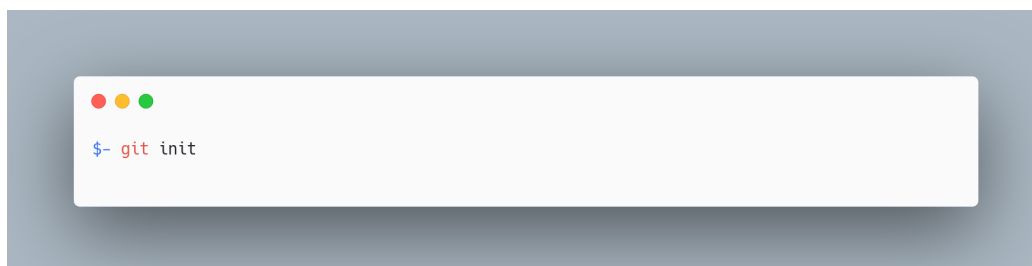
```
package name: (prueba) prueba
version: (1.0.0) 1.0.0
description: Descripcion del paquete
entry point: (index.js) src/index.js
test command:
git repository: https://github.com/FerCorona/crown.git
keywords: React Libreria
author: Fernando Garcia Corona
license: (ISC) MIT
About to write to /Users/fernandogarciacorona/Desktop/prueba/package.json:

{
  "name": "prueba",
  "version": "1.0.0",
  "description": "Descripcion del paquete",
  "main": "src/index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/FerCorona/crown.git"
  },
  "keywords": [
    "React",
    "Libreria"
  ],
  "author": "Fernando Garcia Corona",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/FerCorona/crown/issues"
  },
  "homepage": "https://github.com/FerCorona/crown#readme"
}

Is this OK? (yes) yes
```

### 6.1.3. INICIALIZACIÓN DE GIT EN NUESTRO PROYECTO

Ahora se continuará agregando GIT en nuestro proyecto, esto nos garantizara el control de los cambios que se vayan realizando, para en caso de catástrofes poder regresar a una versión anterior, También podemos crear ramas, para alojar nuevas funcionalidades que se requieran ser agregadas, eso sin afectar el estado del proyecto que ya está funcionando y cuando la nueva función esté completa y probada poder mezclarla con el original (la rama master). Incluir GIT no es una tarea compleja, basta con ejecutar el siguiente comando en la línea de comandos, esto dentro de nuestro directorio (“/Documents/crown”).



Crearé una carpeta oculta (“/.git”) que nos permitirá manipular nuestro código con la línea de comandos de git, como crear ramas, hacer commit, hacer el merge de una rama etc. Para que esto funcione es necesario que el archivo creado anteriormente “package.json” conozca la ubicación remota de nuestro repositorio, se creó una cuenta en GITHUB y se agregó un repositorio, el cual debemos copiar la dirección url y pegarla en el archivo “package.json”, en el apartado llamado “repository” , en la llave “url” como

se muestra en la imagen. Al la url que acabamos de copiar agregamos el prefijo “git+” y el postfijo “.git”.



#### 6.1.4. CONFIGURACIÓN WEBPACK

WEBPACK es una tecnología utilizada en gran cantidad de proyectos de Front-end. Es útil cuando se trabaja en base a una estructura modular, en este caso modula nuestra librería para poder ser agregada en otros proyectos. Nos permite que el resultado final de nuestro proyecto sea menos pesado, esto es logrado por que concatena el código eliminando espacios no necesarios para el intérprete del navegador, lo que deja un archivo con código que no es del todo entendible para las personas pero que es muchos bits menos pesado para el navegador. También nos permite agregar cargadores para que pueda soportar SCSS, HTML JSX imágenes y otros archivos más. Para hacerlo funcionar debemos ejecutar una serie de comandos, que se enlistan a continuación, con ayuda de NPM.



```
$- npm install webpack-cli webpack --save-dev
```

Los anteriores comandos agregan al proyecto en núcleo de WEBPACK, así como su cliente de comandos para la manipulación y visualización de archivos. Después de ejecutar los comandos se debe agregar los siguientes comandos en archivo “package.json” en la sección de scripts, los script son los siguientes.



```
"scripts": {  
  "build": "webpack --mode production",  
  "watch": "webpack --mode production --watch"  
}
```

Los comandos agregados nos son de utilidad para:

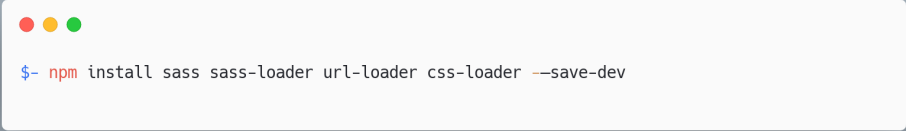
- **Build:** Crear un archivo que estará listo para producción.
- **Watch:** Nos proporciona la misma funcionalidad que Build pero este, puede observar los cambios que estamos haciendo en tiempo real y actualizará el archivo final cada vez.

Para agregar configuración es necesario crear un archivo como se ilustra con el siguiente comando, estando en el directorio raíz de “crown”.



```
$- touch webpack.config.js
```

Necesitamos agregar ciertos cargadores de WEPACK para poder usar SASS y CSS, con el siguiente comando.



```
$- npm install sass sass-loader url-loader css-loader --save-dev
```

Con esto tenemos listas las dependencias necesarias para usar SASS / CSS y para poder manejar archivos como imágenes en JAVASCRIPT, tenemos que agregar la siguiente configuración en nuestro archivo webpack.config.js.

```

const path = require('path');

module.exports = {
  mode: 'production',
  entry: './src/index.js',
  output: {
    path: path.resolve('dist'),
    filename: 'index.js',
    libraryTarget: 'commonjs2'
  },
  module: {
    rules: [
      {
        test: /\.([ac]ss|css)$/i,
        use: [
          'style-loader',
          'css-loader',
          'sass-loader'
        ]
      },
      {
        test: /\.([png|jpg|gif])$/i,
        use: [
          {
            loader: 'url-loader'
          }
        ]
      }
    ]
  },
  resolve: {
    extensions: [ '.js' ]
  }
};

```

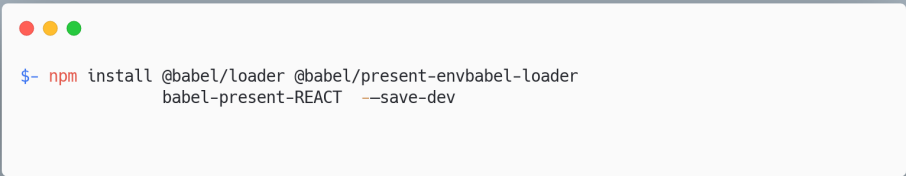
La anterior configuración muestra cómo debe procesar cada tipo de archivo que encuentre dentro de nuestro proyecto, por eso dada una expresión regular que define extensiones de archivos puede usar un cargador a usar. Lo que es definido en la configuración es:

- **Entry:** Es el archivo inicial sobre el cual empezará el análisis del código si este tiene un import de otro archivo continuará sobre ese, esto generará un árbol. El directorio `/src/index.js` fue el dado para este caso.
- **Output:** Es el archivo en el que quedará la salida de nuestra librería en la dirección `/dist/index.js`

- **Rules:** Está incluido dentro de los módulos, esto agrega la manera en cómo se procesarán los archivos SCSS y CSS con las dependencias style-loader, css-loader, sass-loader y por otra parte las imágenes con url-loader.

### 6.1.5. CONFIGURACIÓN BABEL

BABEL es un traductor de código JAVASCRIPT que permite convertir código de nuevas generaciones como el ES6 a versiones antiguas, extendiendo la compatibilidad a navegadores más viejos como Internet Explorer. Solo es necesario agregar las siguientes dependencias, con el siguiente comando.

A terminal window with a light gray background and a dark gray border. It has three colored window control buttons (red, yellow, green) in the top left corner. The command being entered is `npm install @babel/loader @babel/preset-envbabel-loader babel-present-REACT --save-dev`.

```
$- npm install @babel/loader @babel/preset-envbabel-loader
                babel-present-REACT --save-dev
```

Después debemos crear un archivo en el directorio raíz llamado “.babelrc”, al cual debemos agregar la siguiente configuración.

A terminal window with a light gray background and a dark gray border. It has three colored window control buttons (red, yellow, green) in the top left corner. The configuration being entered is a JSON object: `{ "presets": [ "@babel/preset-env", "@babel/preset-REACT" ] }`.

```
{
  "presets": [ "@babel/preset-env", "@babel/preset-REACT" ]
}
```



Y finalmente solo debemos agregar la siguiente configuración al archivo `webpack.config.js` en el apartado de “rules” dentro de “module”.



```
{
  test: /\.jsx?$/,
  exclude: /node_modules/,
  use: {
    loader: 'babel-loader'
  }
}
```

Esto para que WEWPACK sea el encargado de traducir el código con ayuda de BABEL.

### 6.1.6. AGREGAR REACT AL PROYECTO

REACT es una parte fundamental en el desarrollo de la presente librería y para agregarlo es necesario ejecutar el siguiente comando.



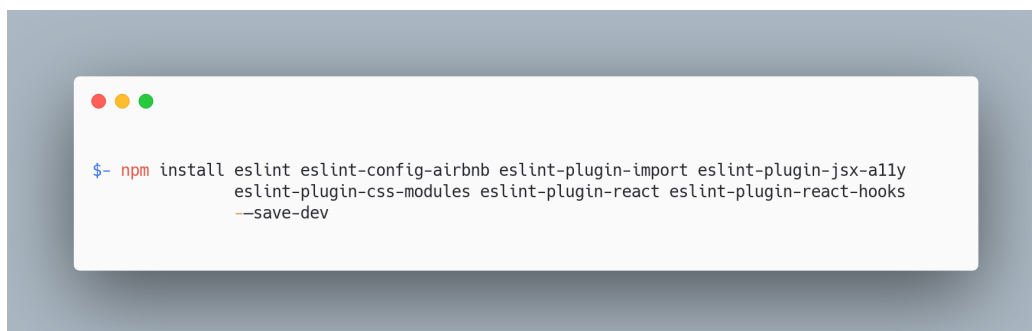
```
$- npm install react
```

Para este comando no se agrega la bandera “--save-dev” al final, por que

de esta manera forzamos a que cuando se instale esta librería también se instale REACT en caso de que no estuviera, ya que nuestra librería necesita REACT para su ejecución.

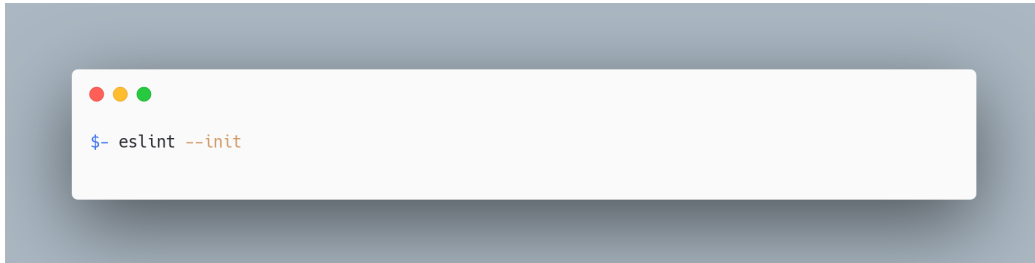
### 6.1.7. CONFIGURACIÓN ESLINT

Finalmente, para que nuestro espacio de desarrollo quede listo agregaremos ESLINT que es un verificador de sintaxis, para tener un código limpio, con una clara indentación. Para que en toda la librería tengamos un código unificado. De igual manera no tendremos que preocuparnos por esto si no que al guardar el archivo obtenga el formato correcto. Para esto debemos ejecutar las siguientes dependencias en la terminal.

A terminal window with a light gray background and a dark gray border. It features three colored window control buttons (red, yellow, green) in the top-left corner. The terminal text is as follows:

```
$- npm install eslint eslint-config-airbnb eslint-plugin-import eslint-plugin-jsx-a11y  
eslint-plugin-css-modules eslint-plugin-react eslint-plugin-react-hooks  
--save-dev
```

Dentro de las dependencias que agregamos se encuentran algunos complementos (plugins) que nos ayudan a dar formato a los archivos JSX, a verificar la sintaxis de REACT y puede verificar que la importación de algún archivo realmente arroje un resultado. Debemos agregar un archivo en el que definiremos un conjunto de reglas específicas las cuales nuestros archivos van a cumplir, para agregar el archivo se debe ejecutar el siguiente comando.



Después de ejecutar el comando este preguntara por datos para la configuración de ESLINT, a continuación, se enlistan cada uno de los datos introducidos.

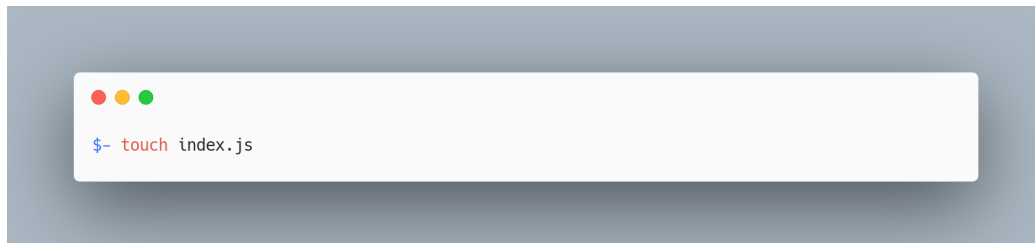
- **Uso de ESLINT:** Seleccionaremos el uso que le daremos a ESLINT, que en este caso es checar sintaxis, encontrar problemas y forzar el estilo del código.
- **Tipo de módulos que usa el proyecto:** Seleccionaremos la opción JAVASCRIPT modules, por que es el tipo de importaciones y exportaciones que estaremos usando.
- **Framework por usar:** Seleccionaremos REACT.
- ¿Se usará TYPESCRIPT?: Se selecciona no.
- ¿Donde se ejecutará?: Debemos seleccionar navegador.
- **Guía de estilos que se usará en el proyecto:** Nosotros elegiremos usar una guía popular y después escogemos AIRBND.
- Formato del archivo de salida: Nosotros debemos escoger JAVASCRIPT.
-

Con esto tenemos todo listo para comenzar con el desarrollo de nuestra librería.

## 6.2. DESARROLLO

### 6.2.1. CONFIGURACIÓN ARCHIVO INICIAL

El proyecto contendrá un nodo raíz el cual será el que tendrá los llamados al conjunto de elementos que tendrá nuestra librería. Anteriormente se creó un directorio llamado “src” en la raíz de “crown”, dentro de “src” debemos crear un archivo llamado “index.js” este es un archivo de JAVASCRIPT el cual definiremos como punto de inicio de webpack y a partir de este buscará todas las importaciones de otros submódulos. Con el siguiente comando creamos el archivo requerido.



Dentro de este archivo debemos poner el llamado a cada elemento que se vaya agregando a nuestra librería ( Botones, Campos de texto, Tablas etc.), en la siguiente ilustración se muestra la manera en que se deben agregar los elementos.



```
import Button from './Button';
import ElementoN from './ElementoN';

export {
  Button,
  ElementoN
};
```

El primer bloque de código muestra la importación de cada uno de los elementos a nuestro archivo index, y la segunda parte exportamos un objeto de JAVASCRIPT con cada uno de los elementos. WEBPACK analiza cada elemento que se incluye en el objeto y busca el contenido existente dentro de cada uno. Cada uno de los elementos que se necesita agregar deben estar situados a la misma altura del archivo “index.js” esto dentro del directorio “src” para que puedan ser procesados.

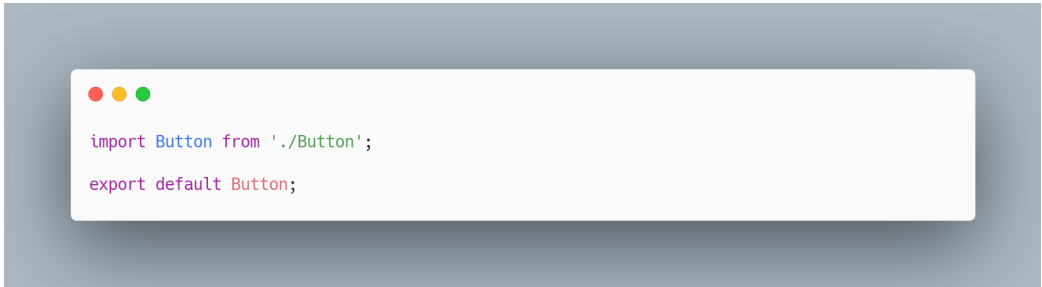
### 6.2.2. ELEMENTO BOTÓN

El primer elemento que vamos a agregar es el botón, para esto creamos un directorio llamado “Botton” de la siguiente manera.



```
$- mkdir Button
```

Y dentro de este directorio crearemos dos archivos que son los que incluirán el núcleo de nuestro botón, el primer archivo se llamará “index.js ” esto es así ya que por defecto, cuando importas un archivo que está dentro de un directorio, JAVASCRIPT toma el que es llamado “index.js” y no es necesario especificar este dato, esto lo podemos ver de manera clara en el archivo inicial “index.js” que está al mismo nivel que el directorio “Botton”, el cual importa el elemento Botton pero no especifica el archivo. Este archivo solamente hace el llamado al segundo archivo “Botton.js” que es el que tiene el código fuente del botón.



```
import Button from './Button';  
export default Button;
```

El segundo archivo ya mencionado “Botton.js” hace el render de nuestro botón, y gestiona la configuración que el usuario final quiera asignarle. En la siguiente tabla se muestran los props ( configuración inicial ) del botón.

Nombre	Uso	Tipo de dato	Valor por defecto
text	Texto que mostrará el botón.	Cadena de texto.	Click me!
onClick	Es la función que ejecutará cuando se hace click.	Funcion.	Función vacía.
color	Es el color del botón.	Cadena de texto.	–blue-4
textColor	Es el color del texto en el botón.	Cadena de texto.	–white
borderColor	Es el color del borde del botón.	Cadena de texto.	–blue-4
type	Es el tipo de botón.	Cadena de texto.	Default
shape	Es la forma del botón. Estos valores agregaran una curvatura, tenemos Round y SemiRound uno mas curvo que otro.	Cadena de texto.	Round
shadow	Especifica si el botón debe tener una sombra.	Cadena de texto.	Booleano.

Los tipos (type) de botones que podemos tener son los siguientes.

- **Default:** Es el botón que se recomienda usar como principal.
- **Secondary:** Es recomendable usarlo si ya se usa un botón default, para



cuidar la jerarquía visual.

- **Text:** Este es un botón sin contorno, debe ser usado para acciones con poca importancia.

Los colores que se usan son una constante definida por la librería ya que se considera que son cromáticamente compatibles entre ellos, este tema se abordará más adelante. A continuación se muestra un fragmento del código que permite agregar la configuración requerida.

```
const Button = ({
  text,
  onClick,
  color,
  textColor,
  borderColor,
  type,
  shape,
  shadow
}) => {
  const style = {
    boxShadow: shadow && '0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19)'
  };
  switch (type) {
    case 'Secondary':
      style.border = `2px solid var(${borderColor})`;
      style.color = `var(${textColor})`;
      break;
    case 'Text':
      style.border = 'none';
      style.color = `var(${textColor})`;
      style.boxShadow = 'none';
      break;
    default:
      style.backgroundColor = `var(${color})`;
      style.color = `var(${textColor})`;
      break;
  };
  return (
    <a
      className={`CROWNButton ${shape}`}
      style={style}
      onClick={onClick}
    >
      {text}
    </a>
  );
};
```

En la primera parte podemos ver una lista de la configuración dada, después definimos un objeto llamado styles el cual agrega dentro de un switch el CSS para que sea mostrado de acuerdo al tipo ( type ) de botón que se quiere. Finalmente regresa HTML el cual es nuestro botón configurado. Se puede observar que dentro de la etiqueta `ja` tenemos “className” esto agrega la clase en la cual definiremos el CSS. Dentro del archivo CSS agregamos el diseño base de nuestro botón, aquí tenemos el tamaño de la letra, tamaño del botón, grosor de letra y otras cosas más.

A code editor window with a light gray background and a dark gray border. It contains CSS code for a class named .CROWNButton. The code is color-coded: property names are in blue, values are in orange, and comments are in red. The code defines font-size, text-align, display, font-weight, width, margin-top, margin-bottom, padding, and cursor for the button class.

```
.CROWNButton {  
  font-size: 14px;  
  text-align: center;  
  display: inline-block;  
  font-weight: 600;  
  width: -webkit-fill-available;  
  margin-top: 5px;  
  margin-bottom: 5px;  
  padding: 15px 30px;  
  cursor: pointer;  
}
```

### 6.2.3. ELEMENTO LABEL

Ahora vamos a agregar el elemento Label que es un texto con el cual tenemos un formato estandarizado para tener un mejor diseño. En este elemento podemos dar parámetros de configuración como el tamaño, grosor y color

que son los elementos básicos que se usan cuando estamos formateando con CSS nuestro texto. A continuación se presenta una lista de los parámetros y su significado.

Nombre	Uso	Tipo de dato	Valor por defecto
text	Este parámetro indica el texto que vamos a mostrar.	Cadena de texto.	"I'm a label"
size	Indicamos el tamaño del texto. Se cuenta con un conjunto de tamaños establecidos que se mencionan en el apartado de las constantes.	Cadena de texto.	small
color	Es el color del texto.	Cadena de texto.	-black-0
weight.	Indicamos el grosor del texto. Se cuenta con un conjunto de tamaños establecidos que se mencionan en el apartado de las constantes.	Cadena de texto.	regular

Para el funcionamiento de la etiqueta de texto se tiene un componente de react en el cual se crea un objeto de JAVASCRIPT, se agrega la configuración del color, tamaño de texto y grosor. Finalmente se crea una etiqueta de HTML a la cual le damos nuestro objeto para que aplique el estilo y le

ponemos el texto que se quiere mostrar.

```
const Label = ({ text, color, size, weight }) => {  
  const style = {  
    color: `var(${color})`,  
    fontSize: `${LABEL_SIZE[size]}`,  
    fontWeight: `${LABEL_WEIGHT[weight]}`  
  };  
  return (  
    <div  
      className='CROWNLabel'  
      style={style}  
    >{text}  
    </div>  
  );  
};
```

#### 6.2.4. ELEMENTO INPUT TEX

El input text es de utilidad para permitir la entrada de datos, y posteriormente puede ser procesado para enviarlos a algún servicio o procesarlos dentro de nuestra aplicación en donde es implementado. Los datos de entrada que son necesarios para configurar el elemento se muestran a continuación.

Nombre	Uso	Tipo de dato	Valor por defecto
placeholder	Es el texto que muestra el InputText si se desea poner.	Cadena de texto.	“Write on me!”
onChange	Es la función que va a hacer invocada cuando se escriba.	Función vacía.	Función vacía.
nameState	Es el nombre con el cual se identifica en el estado.	Cadena de texto.	input
type	Es la manera en como será interpretado por InputText, estos son los que existen por defecto en HTML. Password etc.	Cadena de texto.	text
title	Si se pone como “true” este agregara un Label antes del Input Text y no lo pondrá dentro.	Booleano.	false
extraStyle	Si se desea agregar más estilos se puede poner el nombre de la clase CSS para poder ser manipulado.	Cadena de texto.	Cadena vacía.

La primera consideración que se tiene, es saber si se necesita tener un “Título” en nuestro Input Text, y si es así, se agrega un elemento Label antes, para mostrar cual es el significado de nuestro label. Esto por ejemplo, si se

quiere poner una entrada de texto para solicitar el correo electronico de cierta persona, esto puede estar dentro de el Input Texto denominado placeholder o afuera denominado title, por defecto title está desactivado y se activa enviando “true”. Cuando se escribe sobre el campo de texto, este llama a la función “onChange” que recibe, se regresa el nombre de como está identificado en el estado de REACT y el valor que contiene el campo de texto. Los demás parámetros de los estilos también son agregados en el siguiente código.

```

const InputText = ({
  placeholder,
  value,
  onChange,
  namestate,
  type,
  title,
  extraStyle,
  stateName
}) => {
  let input = null;
  if (title) {
    input = (
      <
        <Label size='small' text={placeholder} color='--black-0' weight='ligh_x' />
        <input
          className={`CROWNInputText ${extraStyle}`}
          type={type}
          value={value}
          onChange={e => onChange({
            stateName,
            value: e.target.value
          })}
          name={namestate}
        />
      </>
    );
  } else {
    input = (
      <input
        placeholder={placeholder}
        className={`CROWNInputText ${extraStyle}`}
        type={type}
        value={value}
        onChange={e => onChange({
          stateName,
          value: e.target.value
        })}
        name={namestate}
      />
    );
  }
  return input;
};

```

**6.2.5. ELEMENTO DROP DOWN**

**6.2.6. ELEMENTO RADIO BUTTON**

**6.2.7. ELEMENTO SWITCH**

**6.2.8. ELEMENTO TABLE**

**6.2.9. ELEMENTO CHECK BOX**

**6.2.10. ELEMENTO IMAGE**

**6.3. TEST**



# RESULTADOS

# **CONCLUSIONES Y RECOMEN- DACIONES**

# FUENTES DE INFORMACIÓN

# ANEXOS