

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>

1 de febrero de 2021

Índice general

Índice de figuras	2
1. Introducción	5
2. Justificación	6
3. Objetivos	11
3.1. Objetivos Generales	11
3.2. Objetivos Específicos	11
4. Caracterización Del Área En La Que Se Participó	12
5. Problemas A Resolver Con Su Respectiva Priorización	13
6. Alcances Y Limitaciones	15
6.1. ALCANCES	15
6.2. LIMITACIONES	17
7. Marco Teórico	18
8. Procedimiento Y Descripción De Las Actividades Realizadas	22
8.1. Creación Del Ambiente De Desarrollo	25
8.1.1. Ambiente De Desarrollo	25
8.1.2. Inicialización Del Archivo NPM	26
8.1.3. Inicialización De Git En Nuestro Proyecto	29
8.1.4. Configuración Web-pack	30

8.1.5.	Configuración Babel	34
8.1.6.	Agregar React Al Proyecto	35
8.1.7.	Configuración ESLint	36
8.2.	Desarrollo	38
8.2.1.	Configuración Archivo Inicial	38
8.2.2.	Elemento Botón	40
8.2.3.	Elemento Label	45
8.2.4.	Elemento Input Tex	47
8.2.5.	Elemento Drop Down	50
8.2.6.	Elemento Radio Button	53
8.2.7.	Elemento Switch	55
8.2.8.	Elemento Table	57
8.2.9.	Elemento CheckBox	58
8.2.10.	Elemento Image	60
8.2.11.	Elemento Loader	62
8.2.12.	Elemento Card	63
8.2.13.	Elemento Modal	65
8.3.	Test	67
9.	Pruebas y Resultados	68
10.	Conclusiones Y Recomendaciones	71
11.	Fuentes de Información	72
12.	Anexos	76

Índice de figuras

7.1. Impresión de texto con React	21
7.2. Impresión de texto una vez el código es convertido	21
8.1. Resultado esperado de la librería	23
8.2. Resultado de la llamada	24
8.3. Actualización del estado	24
8.4. Moverse entre directorios	25
8.5. Crear nuevo directorio	25
8.6. Crear nuevos directorios	26
8.7. Inicializar NPM	26
8.8. Archivo de salida	28
8.9. Inicializar GIT	29
8.10. Agregar repositorio existente	30
8.11. Agregar Webpack	30
8.12. Agregar scripts	31
8.13. Crear archivo vacío	31
8.14. Agregar cargadores de css	32
8.15. Archivo final	33
8.16. Archivo Babel	34
8.17. Configurar Babel	34
8.18. Agregar Babel en Webpack	35
8.19. Agregar React	35
8.20. Agregar ESLint	36

8.21. Inicializar ESLint	37
8.22. Inicializar archivo vacío	38
8.23. Inclusión de los elementos	39
8.24. Crear directorio para el elemento Button	40
8.25. Contenido del elemento Button	40
8.26. Código fuente del elemento Button	43
8.27. Estilos del elemento Button	44
8.28. Código fuente del elemento Label	46
8.29. Código fuente del elemento InputText	49
8.30. Código fuente del elemento DropDown	52
8.31. Código fuente del elemento RadioButton	54
8.32. Código fuente del elemento Switch	56
8.33. Código fuente del elemento Table	57
8.34. Código fuente del elemento CheckBox	59
8.35. Código fuente del elemento Image	61
8.36. Código fuente del elemento Image	62
8.37. Código fuente del elemento Card	64
8.38. Código fuente del elemento Modal	66
9.1. Instalar Create React App	68
9.2. Crear una nueva app	69
9.3. Directorios al crear una nueva app	69
9.4. Hacer el puente de manera local	70
9.5. Usar la librería en la aplicación	70

Capítulo 1

Introducción

Capítulo 2

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 una área de oportunidad para ejercer. 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 año pasado (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 empotrado 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 modulado. 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 desarro-

lladores 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 (css) 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 el paso de elemento. 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.

Capítulo 3

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.

Capítulo 4

Caracterización Del Área En La Que Se Participó

El presente trabajo fue desarrollado para el uso libre y futura implementación en proyectos de software. Específicamente en el ambiente de la programación web, para su utilización en áreas en las cuales se usa como base la librería React, ya que nuestro trabajo funciona sobre React.

Esta librería se encuentra publicada en el gestor de paquetes de JAVASCRIPT llamado NPM.

El rol que toma más partido en el uso de nuestra librería es el Front-end, ya que es el área que desarrolla la parte visual para el software. La librería brinda la facilidad de implementar elementos como botones, imágenes, campos de texto, etiquetas de texto, etc. de una manera simple y sin necesidad de darle un diseño a cada elemento al dar parámetros simples como el como pueden ser color y tamaño.

Capítulo 5

Problemas A Resolver Con Su Respectiva Priorización

La inspiración del presente proyecto nace como la solución al conjunto de problemas que se encontraron al estar desarrollando una página web.

- **Personal**

Al estar desarrollando una página web es posible pertenecer a un equipo en el cual no te tiene un integrante del área de diseño encargado del diseño de interfaz y diseño de experiencia de usuario, el cual brinda el punto de partida para las personas que se encargan de la maquetación de una web, esto puede ser por que no se cuenta con los recursos monetarios para contratar a una persona o por inasistencia.

- **Tiempo**

Existen ocasiones en los desarrollos de software, en los que el tiempo previsto es afectado por resolver otros problemas existentes. En esos casos el tiempo para tener una nueva funcionalidad se reduce o simplemente el tiempo estimado para terminar el trabajo es erróneo. Al tener una librería ganas el tiempo que puedes perder si inicias desde cero. Otras veces se le da más peso a la funcionalidad y se considera un buen diseño y no quieres perder calidad.

- **Estandarización**

Cuando desarrollas una pieza de software, esta cuenta con múltiples colaboradores cada uno con distintas maneras de pensar y por consiguiente tu código tiene múltiples estilos de código. Al usar esta librería se está estandarizando tu trabajo, y cualquier nuevo integrante o cualquier cambio en los roles del equipo puede adaptarse fácilmente.

- **Soporte**

Al consumir los recursos de alguien más, estás garantizando que cualquier error que esto genere no serás el responsable de resolverlo.

- **Reinvención**

Cuando ya existe alguna funcionalidad trabajando correctamente no es necesario desgastarse en volver a generar el trabajo.

Capítulo 6

Aalcances Y Limitaciones

6.1. ALCANCES

Para el desarrollo del presente proyecto se contempla la implementación de un conjunto de elementos, que podrán ser integrados en cualquier proyecto en el que se use JavaScript y el gestor de paquetes NPM. Debe considerarse que al momento de instalar la presente librería instalará forzosamente React.

Se enlistan los primeros elementos que son considerados en el desarrollo. Los cuales son los elementos básicos y que al combinarlos es posible generar interacciones mayores.

- Botón
- Campo de texto
- Tabla
- Etiqueta de texto
- Radio botón
- Switch
- Imagen
- Cuadro de selección

- Radio botón

También se incluyen otros elementos que no son considerados como básicos pero son igualmente utilizados, entre ellos se tienen los siguientes:

- Alertas
- Tarjetas de contenido
- Formulario
- Barra de navegación
- Ventanas modales

Finalmente con una complejidad mayor se tendrán vistas completas que pueden ser usados, estos son:

- Formulario
- Inicio de sesión
- Registro de datos

El conjunto de elementos a desarrollar se tendrán debidamente funcionales y supervisados bajo un marco de pruebas de JavaScript, para esto se estará usando el framework Jest así garantizando la funcionalidad esperada para los usuarios que desean usar la librería, evitando los problemas que pudieran encontrar en los proyectos que lo implementaran. Cada elemento estará debidamente estilizado bajo consideración propia y los conocimientos adquiridos sobre la presente acerca de la experiencia de usuario y el diseño de interfaces.

6.2. LIMITACIONES

Actualmente se considera que gran parte de la planeación inicial pueda ser llevada al ambiente de producción, aunque actualmente ya se encontraron algunas limitaciones derivado al alto tiempo que se tomará durante el desarrollo y la documentación necesaria para los usuarios. Esta limitación no afecta gravemente al desarrollo planeado inicialmente, ya que los elementos básicos se lograran tener listos al final del proyecto y se les asignará una mayor prioridad. Los elementos que se planea terminaran afectados son los siguientes.

- Formulario
- Inicio de sesión
- Registro de datos

Anteriormente se planeaba tener una mayor cantidad de parámetros personalizables a los iniciales, pero esto toma gran cantidad de tiempo. Ahora se considerarán sólo parámetros básicos necesarios para el correcto funcionamiento como el color y los datos a solicitar para el caso de la vista de registro.

Capítulo 7

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 con-

trolador, 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 va reflejado en memoria de una forma más rápida, después el DOM virtual será comparado con el DOM del navegador y solo actualizara 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.



Figura 7.1: Impresión de texto con React

El código será convertido en el que se muestra En la figura 7.2, el cual es interpretada por un mayor número de navegadores.



Figura 7.2: Impresión de texto una vez el código es convertido

- **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.

Capítulo 8

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 flujo de la Figura 8.1.



```

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;

```

Figura 8.1: Resultado esperado de la librería

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 en seguida.



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

Figura 8.2: Resultado de la llamada

- **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 (Figura 8.3) 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.



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

Figura 8.3: Actualización del estado

8.1. Creación Del Ambiente De Desarrollo

8.1.1. 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 comando de la Figura 8.4 es posible cambiar de directorio, ejecutandolo en la terminal.

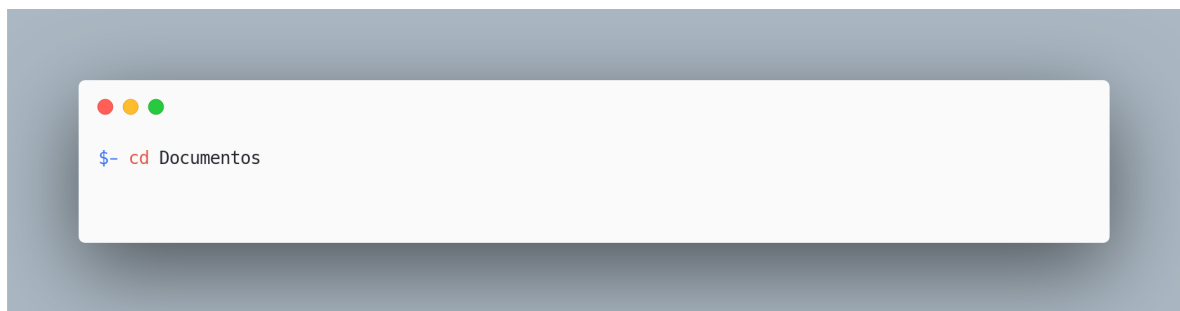


Figura 8.4: Moverse entre directorios

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

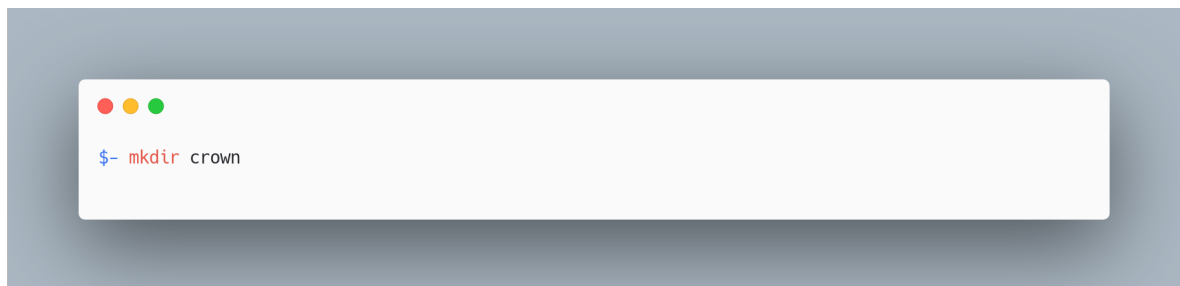


Figura 8.5: Crear nuevo directorio

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, con la ayuda del siguiente comando.

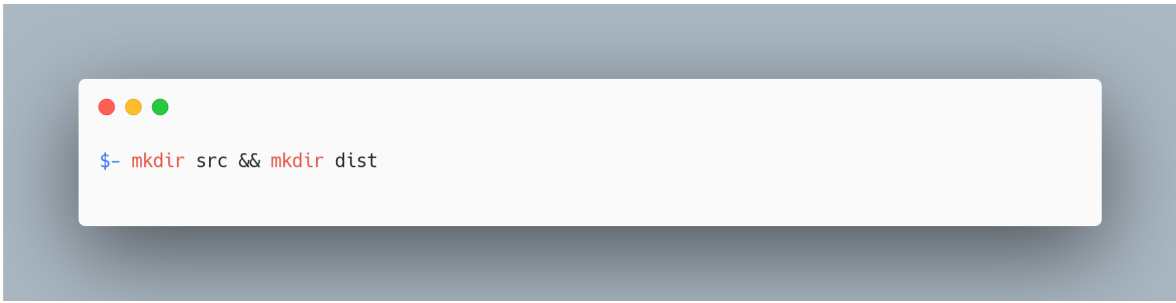


Figura 8.6: Crear nuevos directorios

8.1.2. Inicialización Del Archivo NPM

Se continúa 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.

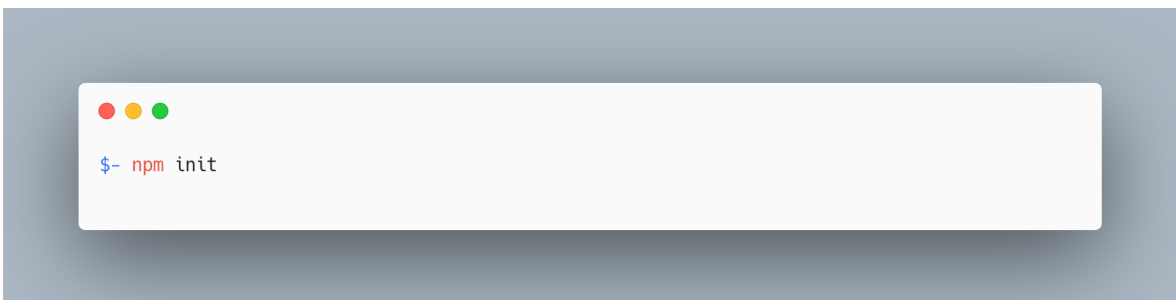


Figura 8.7: Inicializar NPM

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 configuración 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
```

Figura 8.8: Archivo de salida

8.1.3. 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”).

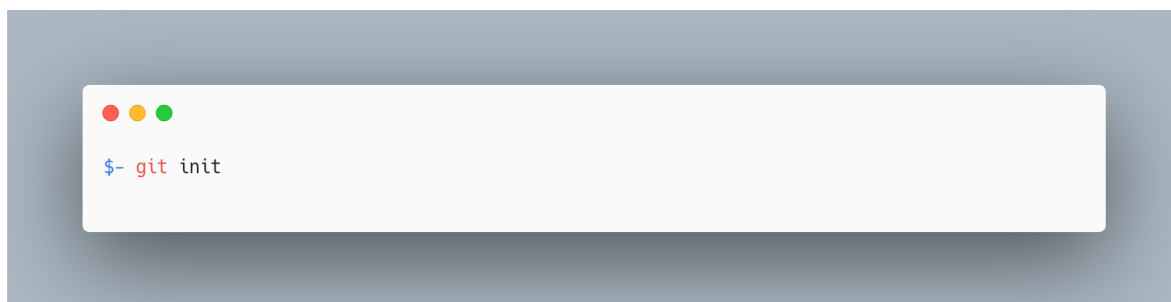


Figura 8.9: Inicializar GIT

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”.



```
"repository": {  
  "type": "git",  
  "url": "git+https://github.com/FerCorona/crown.git"  
}
```

Figura 8.10: Agregar repositorio existente

8.1.4. Configuración Web-pack

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
```

Figura 8.11: Agregar Webpack

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.

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 terminal displays the following JSON snippet:

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

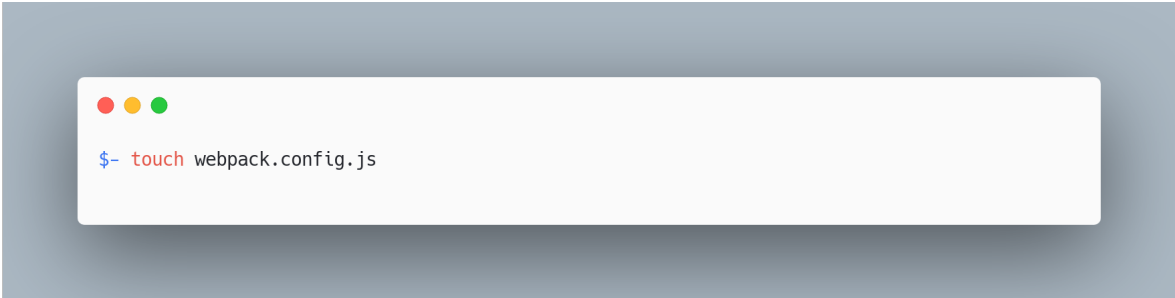
```
"scripts": {  
  "build": "webpack --mode production",  
  "watch": "webpack --mode production --watch"  
}
```

Figura 8.12: Agregar scripts

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”.

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 terminal displays the command to create a new file:

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

Figura 8.13: Crear archivo vacío

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

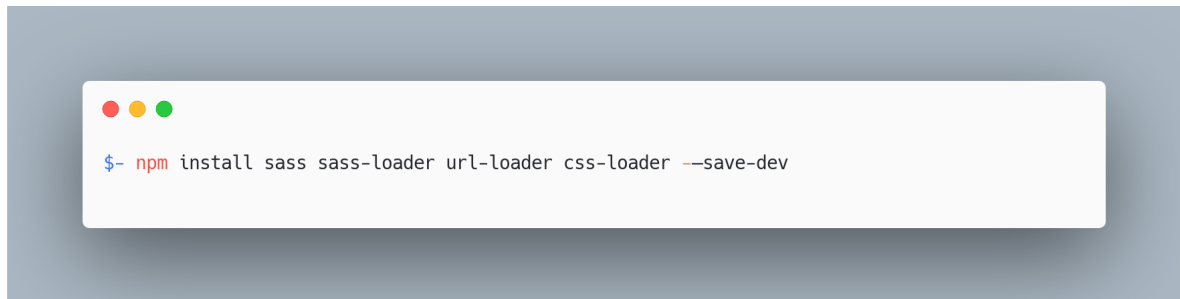


Figura 8.14: Agregar cargadores de css

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' ]
  }
};

```

Figura 8.15: Archivo final

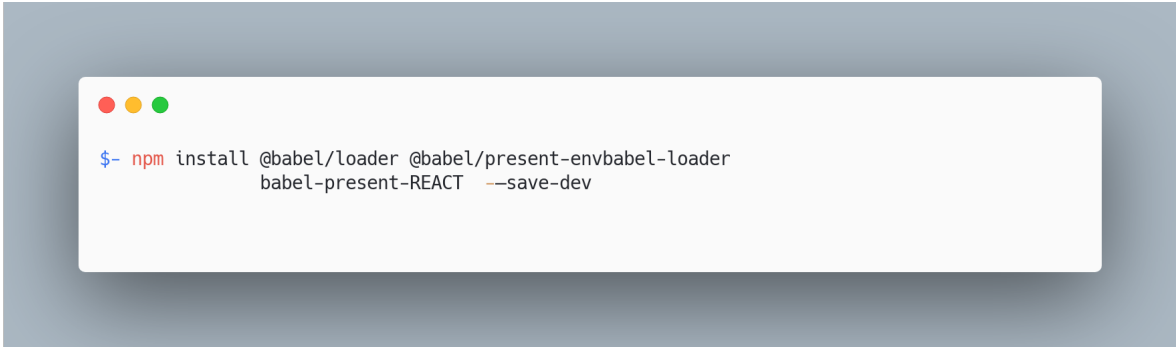
La anterior configuración muestra cómo debe procesar cada tipo de archivo que encuentre dentro de nuestro proyecto, por eso dada una expresión regular que define extensiones de archivos puede usar un cargador a usar. Lo que es definido en la configuración es:

- **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.

8.1.5. 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.



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

Figura 8.16: Archivo Babel

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



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

Figura 8.17: Configurar Babel

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

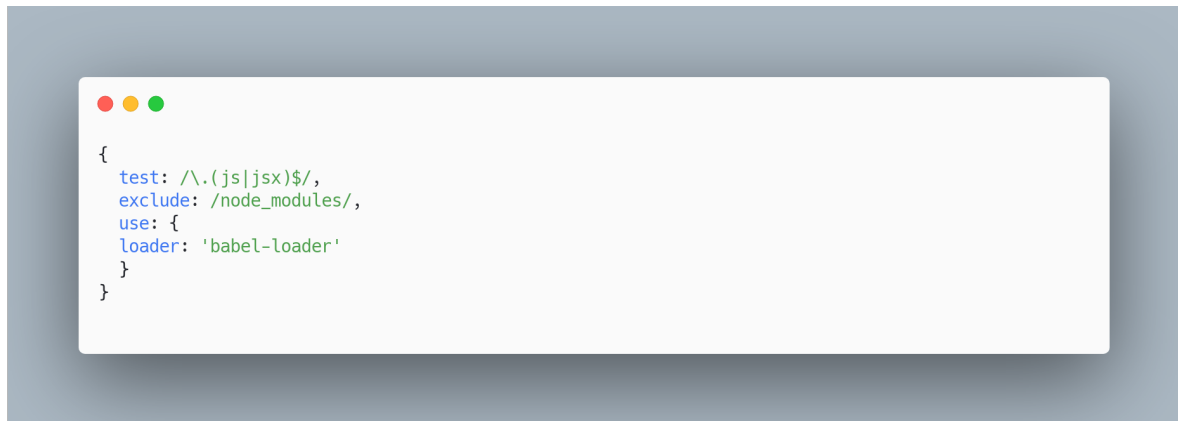


Figura 8.18: Agregar Babel en Webpack

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

8.1.6. 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.

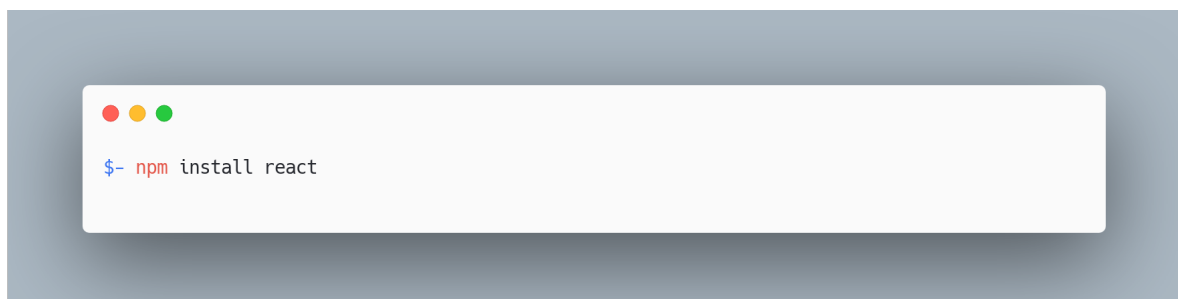


Figura 8.19: Agregar React

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.

8.1.7. 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.

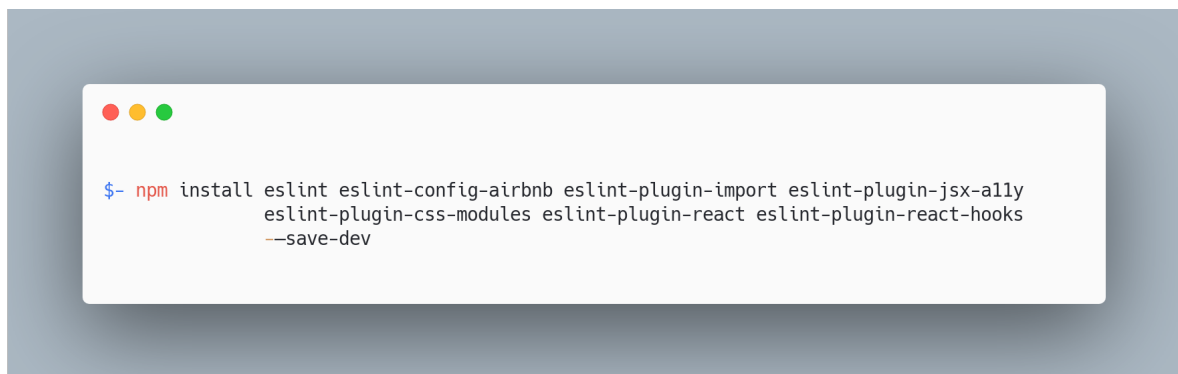


Figura 8.20: Agregar ESLint

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.

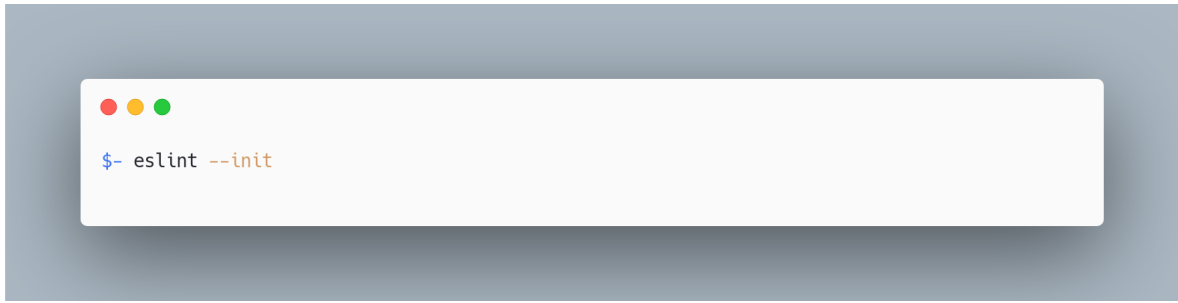


Figura 8.21: Inicializar ESLint

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 Airbnb.
- **Formato del archivo de salida:** Nosotros debemos escoger JavaScript.

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

8.2. Desarrollo

8.2.1. 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.

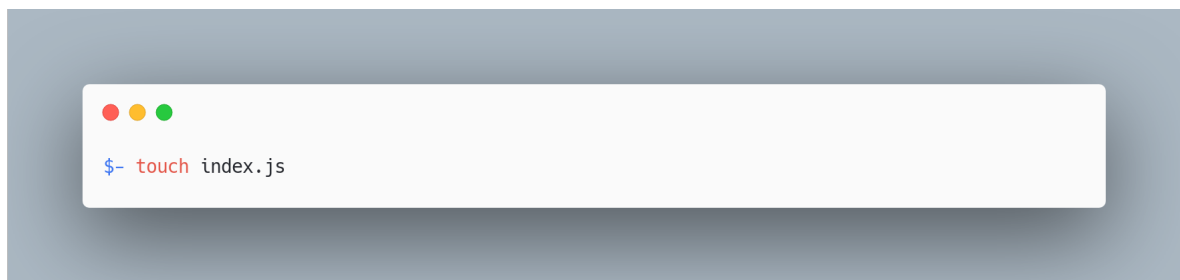


Figura 8.22: Inicializar archivo vacío

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.



Figura 8.23: Inclusión de los elementos

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.

8.2.2. Elemento Botón

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

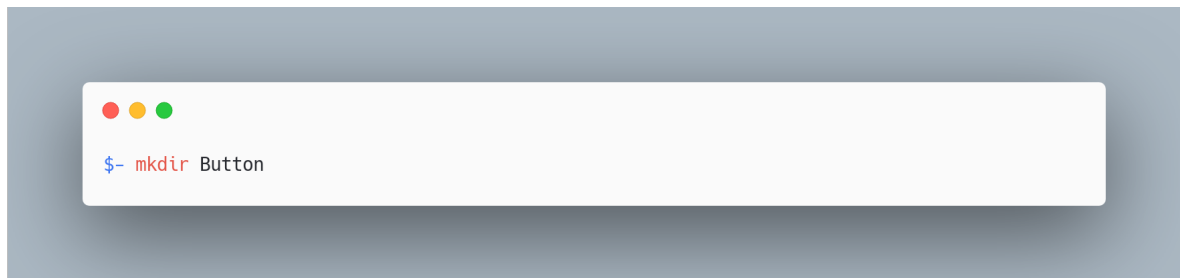


Figura 8.24: Crear directorio para el elemento Button

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 “Button.js” que es el que tiene el código fuente del botón.

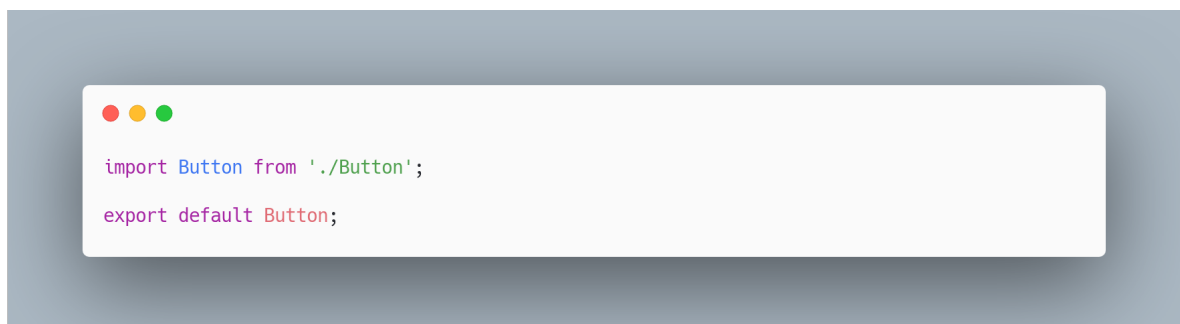


Figura 8.25: Contenido del elemento Button

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 parámetros 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 parámetros obligatorios para su funcionamiento son:

- **text**
- **onClick**

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>
  );
};

```

Figura 8.26: Código fuente del elemento Button

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 ya configurado. Se puede observar que dentro de la etiqueta <a>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.



```
.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;  
}
```

Figura 8.27: Estilos del elemento Button

8.2.3. 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

Los parámetros obligatorios para su funcionamiento son:

- **text**

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>
  );
};
```

Figura 8.28: Código fuente del elemento Label

8.2.4. Elemento Input Text

El input text es de utilidad para permitir la entrada de datos, y posteriormente ser procesados y 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.

Los parámetros obligatorios para su funcionamiento son:

- **placeholder**
- **onChange**
- **nameState**

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 ayudarnos a 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, y 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;
};

```

Figura 8.29: Código fuente del elemento InputText

8.2.5. Elemento Drop Down

Este elemento nos ayuda a permitir la entrada de datos, sin dar el control absoluto al momento de ingresar opciones, dada una lista en la cual es posible seleccionar una de todas. Este elemento está formado por un cuadro, que muestra la opción se encuentra activa (al inicio ninguna), y al hacer click se despliega el total de la lista que se recibe, dando la posibilidad de seleccionar alguna entrada. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
options	Es el conjunto de las opciones para desplegar.	Arreglo de cadenas de texto.	
optionSelected	Es la opción que está seleccionada actualmente.	Cadena de texto.	
onChange	Es una función que se ejecutará cuando se seleccione una opción.	Función.	Funcion vacia.
stateName	Es el nombre como es identificado en el estado de React.	Cadena de texto	

Los parámetros obligatorios para su funcionamiento son:

- **options**
- **optionSelected**
- **onChange**
- **stateName**

El código que se muestra a continuación es el usado para hacer funcionar nuestro elemento, el cual tiene una función que activa y desactiva las opciones de nuestro elemento cuando se hace click sobre este. La acción desencadenada modifica el estado para hacer el cambio.



```

class DropDown extends Component {
  constructor() {
    super();
    this.state = {
      isActive: false
    };
    this._toggleActive = this._toggleActive.bind(this);
  }

  _toggleActive() {
    this.setState({ isActive: !this.state.isActive });
  }

  render() {
    const { optionSelected, options, onChange, stateName } = this.props;
    return (
      <div className='CROWNDropDown'>
        <div className='Options' onClick={() => this._toggleActive()}>
          {optionSelected}
        </div>
        {
          this.state.isActive && (
            <div className='ListContent'>
              {
                options.map((option, index) => (
                  <div
                    className='Option'
                    key={`option-${index}`}
                    onClick={() => this.setState({ isActive: false },
                      onChange({
                        stateName,
                        value: option
                      })
                    )})
                >
                  {option}
                </div>
              )
            </div>
          )
        }
      </div>
    );
  }
}

```

Figura 8.30: Código fuente del elemento DropDown

8.2.6. Elemento Radio Button

Este elemento es usado cuando se quiere dar a elegir al usuario entre un número limitado de opciones y no recomendado mayor a 3, como por ejemplo en un reactivo de un examen cuando se quiere dar a elegir si la afirmación dada es correcta o incorrecta, y las posibles respuestas es Sí o No. Este elemento solo nos permitirá a elegir una opción entre el total de respuestas. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
options	Son las opciones posibles a seleccionar.	Arreglo de Objetos.	
selectedOption	Es la opción que está seleccionada actualmente.	Cadena de texto.	
onChange	Es una función que se ejecutará cuando se seleccione una opción.	Función.	Funcion vacia.
stateName	Es el nombre como es identificado en el estado de React.	Cadena de texto	

Los parámetros obligatorios para su funcionamiento son:

- **options**
- **selectedOption**
- **onChange**
- **stateName**

En el siguiente código se muestra el código fuente del elemento, el cual cuando se activa una opción actualiza el estado de React

```
const RadioButton = ({ options, selectedOption, onChange, stateName }) => (  
  <div className='CROWNRadioButton' role='radiogroup' aria-  
labelledby='bulgy-radios-label'>  
    {  
      options.map(option => (  
        <label>  
          <input  
            type='radio'  
            value={option.id}  
            checked={selectedOption === option.id}  
            onChange={() => onChange({  
              stateName,  
              value: option.id  
            })}  
          />  
          <span className='Radio' />  
          <div className='Label'>  
            <Label size='small' text={option.label} color='--black-0'  
weight='ligh_x' />  
          </div>  
        </label>  
      )  
    )  
  }  
  </div>  
);
```

Figura 8.31: Código fuente del elemento RadioButton

8.2.7. Elemento Switch

Este elemento es recomendable cuando se desea iterar entre dos estados, estos son activo o inactivo. Puede ser usado, por ejemplo, cuando en una web se desea saber si el usuario quiere recibir las últimas noticias generadas. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
isChecked	Indica si el elemento se encuentra activo o inactivo.	Booleano.	
onChange	Es una función que se ejecutará cuando se seleccione una opción.	Función.	Función vacía.
text	Es la etiqueta o texto que acompaña al elemento para indicar su funcionalidad.	Cadena de texto.	Sin texto.
stateName	Es el nombre como es identificado en el estado de React.	Cadena de texto.	

Los parámetros obligatorios para su funcionamiento son:

- **isChecked**
- **onChange**
- **stateName**


```

const Switch = ({ isChecked, onChange, text, stateName }) => {
  const id = uuidv4();
  return (
    <div className='CROWNSwitch'>
      <input
        type='checkbox'
        id={id}
        className='checkbox colorSwitch'
        defaultChecked={isChecked}
        onChange={() => onChange({
          stateName,
          value: !isChecked
        })}
      />
      <label htmlFor={id} className='switch colorSwitch' />
      {
        text && (
          <div className='Label'>
            <Label size='small' text={text} color='--black-0'
              weight='light_x' />
          </div>
        )
      }
    </div>
  );
};

```

Figura 8.32: Código fuente del elemento Switch

8.2.8. Elemento Table

Este elemento es de utilidad para mostrar datos de forma organizada como su nombre lo dice, una tabla. El elemento contiene una cabecera que indica el tipo de dato, nombre etc de la columna. También tiene el cuerpo que es en sí todos los datos. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
header	Es el título de cada columna.	Arreglo de cadenas.	
body	Es el conjunto de datos.	Arreglo de arreglo de cadenas.	Función vacía.

Los parámetros obligatorios para su funcionamiento son:

- **body**



```
const Table = ({ header, body }) => (  
  <table className='CROWNTable'>  
    <thead>  
      <tr className='Header'>  
        {header.map(head => (  
          <th>{head}</th>  
        ))}  
      </tr>  
    </thead>  
    <tbody>  
      {body.map(row => (  
        <tr className='Body'>  
          {row.map(cell => (  
            <td>{cell}</td>  
          ))}  
        </tr>  
      ))}  
    </tbody>  
  </table>  
);
```

Figura 8.33: Código fuente del elemento Table

8.2.9. Elemento CheckBox

Este elemento es recomendable cuando se desea tener una casilla de verificación, puede ser seleccionado o sin seleccionar. Puede ser usado, por ejemplo, cuando en una web se debe forzar a aceptar los términos y condiciones para poder realizar un registro. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
isChecked	Indica si el elemento se encuentra activo o inactivo.	Booleano.	
onChange	Es una función que se ejecutará cuando se seleccione una opción.	Función.	Función vacía.
text	Es la etiqueta o texto que acompaña al elemento para indicar su funcionalidad.	Cadena de texto.	No texto.
stateName	Es el nombre como es identificado en el estado de React.	Cadena de texto.	

Los parámetros obligatorios para su funcionamiento son:

- **isChecked**
- **onChange**
- **stateName**



```

const CheckBox = ({ text, onChange, isChecked, stateName }) => (
  <div className='CROWNCheckBox Active'>
    <input
      id={uuidv4()}
      type='checkbox'
      className='Active'
      onChange={() => onChange({
        stateName,
        value: !isChecked
      })}
      checked={isChecked}
    />
    {
      text && (
        <div className='Label'>
          <Label size='small' text={text} color='--black-0' weight='ligh_x' />
        </div>
      )
    }
  </div>
);

```

Figura 8.34: Código fuente del elemento CheckBox

8.2.10. Elemento Image

Este elemento es de utilidad para agregar una imagen, en el cual solo basta con darle la dirección donde se encuentra ubicada y la librería agrega los estilos necesarios para hacer la imagen lucir. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
src	Es la ubicación de la imagen.	Cadena de texto.	
frame	Es un cuadro que se agregara al contorno de la imagen con un efecto difuminado.	Booleano.	true
shadow	Es una sombra al contorno de la imagen.	Booleano.	true
size	Es el tamaño en pixeles de la imagen.	Arreglo de arreglo de cadenas.	Objeto.

Los parámetros obligatorios para su funcionamiento son:

- **src**

```

const Image = ({ src, frame, shadow, size }) => {
  const backgroundImage = src ? `url(${src})` : 'linear-gradient(45deg,
var(--red-1), var(--pink-1))';
  const onlyImage = (
    <div
      className='Image'
      style={
        {
          backgroundImage,
          backgroundPosition: 'center',
          backgroundRepeat: 'no-repeat',
          backgroundSize: 'cover',
          position: 'relative'
        }
      }
    >
  );
  if (!frame) {
    return (
      <div className={`CROWNImage ${shadow ? 'Shadow' : ''}`} style={size}>
        {onlyImage}
      </div>
    );
  }
  return (
    <div className={`CROWNImage ${shadow ? 'Shadow' : ''}`} style={size}>
      <div
        className='ImageBlur'
        style={
          {
            backgroundImage,
            backgroundPosition: 'center',
            backgroundRepeat: 'no-repeat',
            backgroundSize: 'cover',
            position: 'relative'
          }
        }
      >
        <div className='Blur'>
          {onlyImage}
        </div>
      </div>
    </div>
  );
};

```

Figura 8.35: Código fuente del elemento Image

8.2.11. Elemento Loader

Este elemento se recomienda usar cuando se desea dar la retroalimentación al usuario, cuando se están ejecutando procesos de la aplicación y es necesario esperar a que estos terminen para poder avanzar. Con la presente librería es relativamente simple hacer la implementación.

El parámetro que el elemento necesita para su funcionamiento es.

Nombre	Uso	Tipo de dato	Valor por defecto
show	Indicar si el elemento debe mostrarse.	Booleano.	false

Los parámetros obligatorios para su funcionamiento son:

- **show**



Figura 8.36: Código fuente del elemento Image

8.2.12. Elemento Card

Este elemento se usa cuando queremos mostrar tarjetas de información, las cuales contienen imagen, texto y botones. Por ejemplo si se desea desarrollar una web para un restaurante podemos implementar este elemento el cual nos facilitará la tarea, porque podemos darle la imagen del restaurante o algún platillo que se quisiera mostrar, también le podemos dar la descripción, título y un botón que realizará una acción con cada tarjeta. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
src	Imagen a mostrar.	Cadena de texto.	false
title	Título de la tarjeta.	Cadena de texto.	Card Title
content	Contenido de la tarjeta.	Cadena de texto.	Card Content
buttonConfig	Configuración del botón.	Objeto.	false
customButton	Elemento Button previamente configurado.	Componente.	false

Los parámetros obligatorios para su funcionamiento son:

- **content**
- **buttonConfig**


```
const Card = ({ src, title, content, buttonConfig, customButton }) => (  
  <div className='CROWNCARD'>  
    <div  
      className='Card'  
      style={  
        {  
          backgroundImage: `url(${src})`,  
          backgroundPosition: 'center',  
          backgroundRepeat: 'no-repeat',  
          backgroundSize: 'cover',  
          position: 'relative'  
        }  
      }>  
      <div className='Content'>  
        <div className='Title'>{title}</div>  
        <div className='ContentCard'>{content}</div>  
        <div className='Others'>{customButton ? customButton : <Button  
      {...buttonConfig} />}</div>  
      </div>  
    </div>  
  );
```

Figura 8.37: Código fuente del elemento Card

8.2.13. Elemento Modal

Este elemento es crucial para alertar de una acción la cual requiere de una mayor precaución. Por ejemplo un botón que al presionar ejecuta un proceso el cual elimina algún registro de la base de datos permanentemente. Con este podemos mandar un mensaje de advertencia el cual confirmará o denegará la acción, y luego el flujo continuará habitualmente. Los parámetros que el elemento necesita para su funcionamiento son los siguientes.

Nombre	Uso	Tipo de dato	Valor por defecto
show	Indicar si el elemento debe mostrarse.	Booleano.	false.
onClose	Acción a realizar si el modal se cierra.	Función.	Función vacía.
onActionAccepted	Acción a realizar si se acepta la acción.	Función.	Función vacía.
onActionRejected	Acción a realizar si se deniega la acción.	Función.	Función vacía.
child	Contenido para un modal personalizado.	Componente de React.	Nulo
title	Título del modal.	Cadena de texto.	Modal Title
content	Contenido del modal.	Cadena de texto.	Content Title

Los parámetros obligatorios para su funcionamiento son:

- **show**
- **onClose**

- onChange
- onActionAccepted
- onActionRejected
- content

```
const Modal = ({ show, onClose, onActionAccepted, onActionRejected, child, title, content }) => (
  show && (
    <div className='CROWNModal'>
      <div className='ContainerModal'>
        <div className='Modal'>
          {
            child !== null ? child : (
              <>
                <div className='Close' onClick={() => onClose()}>X</div>
                <div className='Content'>
                  <div className='Title'>{title}</div>
                  <div className='Text'>{content}</div>
                  <div className='ButtonArea'>
                    <Button
                      shape='Square'
                      color='--red-1'
                      text='Calcelar'
                      type='Secondary'
                      borderColor='--red-1'
                      textColor='--red-1'
                      onClick={() => onActionAccepted()} />
                    <Button
                      shape='Square'
                      color='--teal-1'
                      text='Aceptar'
                      onClick={() => onActionRejected()} />
                  </div>
                </div>
              </div>
            )
          }
        </div>
      </div>
    </div>
  )
);
```

Figura 8.38: Código fuente del elemento Modal

8.3. Test

Capítulo 9

Pruebas y Resultados

Con el desarrollo de la librería ya es posible hacer la implementación de un proyecto el cual ya usa React. A continuación se ejemplifica el uso de la librería paso a paso.

Existe una herramienta que nos permite crear el esqueleto de una web basada en React. [7] esta es desarrollada y mantenida por Facebook, de la cual partiremos para crear una web en la que probaremos la librería.

Instalaremos la herramienta llamada Create React App con el siguiente comando.

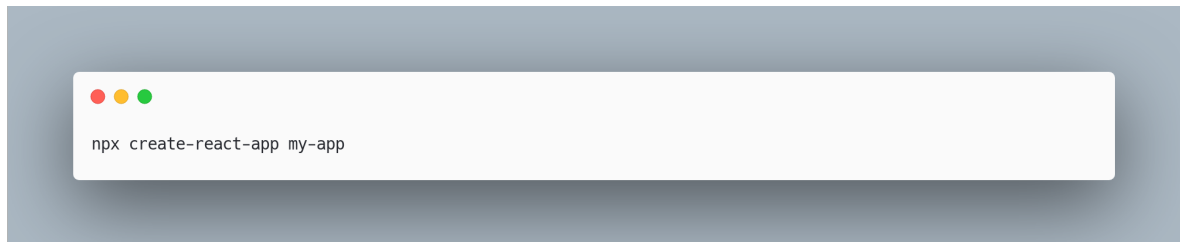


Figura 9.1: Instalar Create React App

Crearemos la web llamada ejemplo-1 con el siguiente comando.



Figura 9.2: Crear una nueva app

El comando nos generará los siguientes directorios.

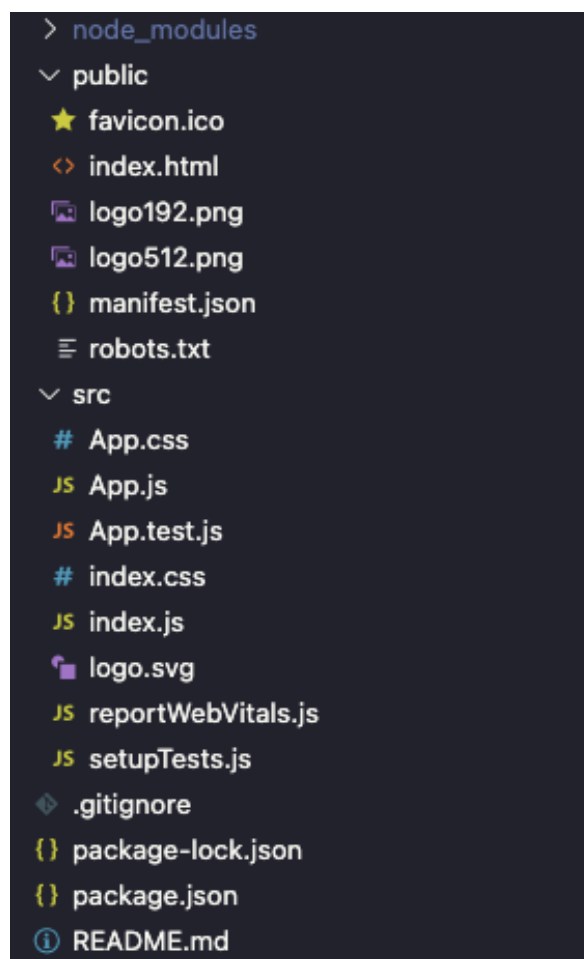


Figura 9.3: Directorios al crear una nueva app

Para poder probar nuestra librería debemos a indicar a la app ejemplo-1 que use la dependencia desde el modo local, esto con el siguiente comando.




```
npm link && npm run watch
```

Figura 9.4: Hacer el puente de manera local

Con esto ya es posible usar la librería en cualquier proyecto local.

Dentro del proyecto ejemplo-1 debemos ejecutar el siguiente comando para incluirlo.



```
npm link crown
```

Figura 9.5: Usar la librería en la aplicación

Ahora basta incluirlo como se agrega cualquier dependencia de node, basta con agregar cualquiera de los elementos aquí desarrollados. El siguiente código incluye en la primera línea, la importación del componente Label, si se quisieran agregar más, debe enlistarse espaciado por una coma. Y en la parte del return se agrega el componente. Este elemento desplegará un texto por defecto en el navegador.



```
import { Label } from 'crown';  
  
function App() {  
  return (  
    <Label />  
  );  
};
```

Figura 9.6: Implementación la librería

Capítulo 10

Conclusiones Y Recomendaciones

Capítulo 11

Fuentes de Información

Bibliografía

- [1] Banks, A., Porcello, E. (2017). Learning REACT Functional Web Development with REACT and Redux. O'Reilly.
- [2] Coyier, C. (2013). SASS FOR WEB DESIGNERS. <https://abookapart.com/products/sass-for-web-designers>
- [3] Danny Goodman. (1998). Dynamic HTML The Definitive Reference. 101 Morris Street, Sebastopol, CA 95472: O'REILLY.
- [4] Fu, C. (2016). Exploration of Web front-end development technology and optimization direction.
- [5] Gaikovina Kula, R., Ouni, A., M. German, D., Inoue, K. (2017, septiembre). On the Impact of Micro-Packages: An Empirical Study of the npm JAVASCRIPT Ecosystem. Osaka University, Japan. <https://arxiv.org/abs/1709.04638>
- [6] Gallegos, G. (2017, 20 octubre). Para que es Webpack? Medium. <https://medium.com/tecninja/porque-usar-webpack-4a5004094455>
- [7] Getting Started | Create React App. (s. f.). Create React App. <https://create-react-app.dev/docs/getting-started/>
- [8] Hudson, P. (2016). Hacking with REACT [Libro electrónico]. <http://www.hackingwithREACT.com/>
- [9] JAVASCRIPT. (2020, 11 agosto). Documentación web de MDN. <https://developer.mozilla.org/es/docs/Web/JAVASCRIPT>

- [10] K. White, S. (2020, 10 enero). The 10 most in-demand tech jobs for 2020 — and how to hire for them. CIO. <https://www.cio.com/article/3235944/hiring-the-most-in-demand-tech-jobs-for-2018.html>
- [11] Mobile-first indexing best practices. (s. f.). Google Developers. <https://developers.google.com/search/mobile-sites/mobile-first-indexing>
- [12] Open Source Initiative. (2006). The MIT license. 2015b.[Online]. Available: <https://opensource.org/licenses/MIT>. [Accessed 27 March 2017].
- [13] Popoter Pérez, G. J. (2016). Rediseño de aplicaciones utilizando las tecnologías modernas para el desarrollo web en su parte Front-end.
- [14] Pozas, J. L. B. (2018, 11 enero). Las 10 profesiones de TI con mayor demanda en 2017. CIO MX. <https://cio.com.mx/las-10-profesiones-ti-mayor-demanda-en-2017/>
- [15] ¿Qué es el open source? (2020). Red Hat. <https://www.redhat.com/es/topics/open-source/what-is-open-source>
- [16] REACT – A JAVASCRIPT library for building user interfaces. (s. f.). REACT. <https://reactjs.org>
- [17] Riquelme, R. (2016, 17 octubre). Los 10 mejores trabajos en TI e Ingeniería. El Economista. <https://www.eleconomista.com.mx/tecnologia/Los-10-mejores-trabajos-en-TI-e-Ingenieria-20161017-0053.html>
- [18] Ruiz Inoue, K. (2020, 20 agosto). ¿Qué es el Desarrollo Front end, Back end y Fullstack? Medium. <https://medium.com/deuk/qué-es-el-desarrollo-front-end-back-end-y-fullstack-2941b51afd1>
- [19] Stack Overflow Developer Survey 2020. (2020). Stack Overflow. <https://insights.stackoverflow.com/survey/2020overview>

- [20] The Linux Foundation. (2018). Enterprise Open Source: A Practical Introduction [Libro electrónico].<https://www.linuxfoundation.org/press-release/2018/08/free-ebook-enterprise-open-source-a-practical-introduction-teaches-enterprises-how-to-accelerate-open-source-efforts/>
- [21] What is npm. (2020). w3schools. https://www.w3schools.com/whatis/whatis_npm.asp
- [22] What is Babel? · Babel. (2020). Babel. <https://babeljs.io/docs/en/>

Capítulo 12

Anexos