



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO



INSTITUTO TECNOLÓGICO DE MORELIA  
"José María Morelos y Pavón"

## **INSTITUTO TECNOLÓGICO DE MORELIA**

**DIVISIÓN DE ESTUDIOS PROFESIONALES**

**DEPARTAMENTO DE SISTEMAS Y  
COMPUTACIÓN**

**TITULACIÓN INTEGRAL POR PROYECTO DE  
INVESTIGACIÓN**

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

**QUE PARA OBTENER EL TÍTULO DE:**

**Ingeniero en Sistemas Computacionales**

**PRESENTA:**

**Fernando García Corona**

**ASESOR:**

**Rogelio Ferreira Escutia**

**MORELIA, MICHOACÁN**

**Marzo del 2021**

**Copia del Oficio de Impresión  
definitiva División de Estudios**

# **Copia del Oficio de Impresión definitiva Sinodales**

# Agradecimientos

# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. Justificación</b>	<b>3</b>
<b>3. Objetivos</b>	<b>8</b>
3.1. Objetivos Generales . . . . .	8
3.2. Objetivos Específicos . . . . .	8
<b>4. Caracterización Del Área En La Que Se Participó</b>	<b>10</b>
<b>5. Problemas A Resolver Con Su Respectiva Priorización</b>	<b>11</b>
<b>6. Alcances Y Limitaciones</b>	<b>13</b>
6.1. Alcances . . . . .	13
6.2. Limitaciones . . . . .	14
<b>7. Marco Teórico</b>	<b>16</b>
<b>8. Procedimiento Y Descripción De Las Actividades Realizadas</b>	<b>23</b>
8.1. Creación Del Ambiente De Desarrollo . . . . .	26
8.1.1. Ambiente De Desarrollo . . . . .	26
8.1.2. Inicialización Del Archivo NPM . . . . .	27
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 . . . . .	35
8.2. Lineamientos UX/UI . . . . .	38
8.3. Desarrollo . . . . .	39
8.3.1. Configuración Archivo Inicial . . . . .	39
8.3.2. Elemento Botón . . . . .	41
8.3.3. Elemento Label . . . . .	45
8.3.4. Elemento Input Tex . . . . .	47
8.3.5. Elemento Drop Down . . . . .	50
8.3.6. Elemento Radio Button . . . . .	53
8.3.7. Elemento Switch . . . . .	55
8.3.8. Elemento Table . . . . .	57
8.3.9. Elemento CheckBox . . . . .	58
8.3.10. Elemento Image . . . . .	60
8.3.11. Elemento Loader . . . . .	62
8.3.12. Elemento Card . . . . .	63
8.3.13. Elemento Modal . . . . .	65
8.4. Test . . . . .	67
8.5. Publicación de la librería en NPM . . . . .	69
8.6. Diferencia entre el uso nativo de React y esta librería . . . . .	70
<b>9. Pruebas y Resultados</b>	<b>75</b>
9.1. Comparación contra Material Design . . . . .	85
<b>10. Conclusiones Y Recomendaciones</b>	<b>87</b>
<b>11. Fuentes de Información</b>	<b>88</b>
<b>12. Anexos</b>	<b>91</b>

# Índice de figuras

7.1. Componente calculadora . . . . .	19
7.2. Llamado a componente . . . . .	20
7.3. Impresión de texto con React . . . . .	22
7.4. Impresión de texto una vez el código es convertido . . . . .	22
8.1. Resultado esperado de la biblioteca . . . . .	24
8.2. Resultado de la llamada . . . . .	25
8.3. Actualización del estado . . . . .	25
8.4. Moverse entre directorios . . . . .	26
8.5. Crear nuevo directorio . . . . .	26
8.6. Crear nuevos directorios . . . . .	27
8.7. Inicializar NPM . . . . .	27
8.8. Archivo de salida . . . . .	29
8.9. Inicializar GIT . . . . .	30
8.10. Agregar repositorio existente . . . . .	30
8.11. Agregar Webpack . . . . .	31
8.12. Agregar scripts . . . . .	31
8.13. Crear archivo vacío . . . . .	32
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 . . . . .	36
8.22. Inicializar archivo vacío . . . . .	39
8.23. Inclusión de los elementos . . . . .	40
8.24. Crear directorio para el elemento Button . . . . .	41
8.25. Contenido del elemento Button . . . . .	41
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
8.39. Instalar dependencias . . . . .	67
8.40. Crear directorio . . . . .	67
8.41. Test del componente Button . . . . .	68
8.42. Agregar comando . . . . .	68
8.43. ingresar a nuestra cuenta . . . . .	69
8.44. Publicar paquete . . . . .	69
8.45. Crear nuevos directorios . . . . .	70
8.46. Crear nuevos directorios . . . . .	71
8.47. Crear nuevos directorios . . . . .	72

8.48. Crear nuevos directorios . . . . .	73
8.49. Crear nuevos directorios . . . . .	73
9.1. Instalar Create React App . . . . .	75
9.2. Crear una nueva app . . . . .	75
9.3. Directorios al crear una nueva app . . . . .	76
9.4. Instalando nuestra biblioteca . . . . .	76
9.5. Implementación la biblioteca . . . . .	77
9.6. Agregar campos al estado de React . . . . .	77
9.7. Código para maquetar . . . . .	78
9.8. Función para actualizar el estado de React . . . . .	79
9.9. Resultado final . . . . .	80
9.10. Resultado final al agregar usuarios . . . . .	81
9.11. Filtro en el estado de React . . . . .	82
9.12. Código para mostrar los platillos . . . . .	82
9.13. Filtrar por comida . . . . .	83
9.14. Filtrar por bebida . . . . .	84
9.15. Confirmación de pedido . . . . .	85

# Capítulo 1

## Introducción

JavaScript cuenta con un gestor de paquetes llamado NPM, este es encargado de gestionar el contenido, cada uno de los paquetes son implementados en proyectos Web. Actualmente no es común encontrar alguna página Web en el que el cien por ciento sea desarrollado por un solo equipo, ya que existen herramientas que son fáciles de agregar y no tienen costo alguno. NPM es el conjunto de paquetes que los desarrolladores desean proveer a la sociedad para su libre uso. Es por esta razón que durante el presente documento se desea desarrollar otra herramienta para que las personas, tengan otra alternativa fácil de usar y que puedan ser implementada en sus proyectos rápidamente. Durante el presente documento se describe el proceso para desarrollar una biblioteca. Se cubrirá la creación del ambiente requerido, instalación de paquetes, creación de ficheros y configuración de los mismos. También se describirá el desarrollo de cada uno de los componentes que se planea tener, se mostrará parte del código fuente que hace que estos funcionen correctamente, y los requisitos necesarios para que trabajen de manera adecuada. Se dará una pequeña introducción al marco de trabajo React, ya que la presente biblioteca trabaja en conjunto con él. En el desarrollo de la biblioteca se implementara una guía de estilos de código para que el código sea un estándar, esto se logra usando la guía de estilos propuesta por Airbnb [1] que es una de las más usadas ya que actualmente obtendremos como resultado final un código fuente consistente en todo el desarrollo del proyecto, también nos ayuda a tener un código mucho más legible y en la medida de lo posible eficiente, aunque en algunos casos nos vemos restringidos al tener un estilo de programación que difiere , así cualquier persona

interesada podrá colaborar para aumentar las funcionalidades una vez una primera versión aquí desarrollada esté lista. Al terminar el desarrollo se tomará el tema de los test que trata de la manera en que se comprueba el correcto funcionamiento de lo aquí hecho. Finalmente se muestra la manera de usar los componentes aquí desarrollados, la manera de instalar la biblioteca y la correcta manera de implementarla.

Esta biblioteca está creada con base en React, motivo por el cual se crean componentes como campos de texto, imágenes, tablas y varios más elementos que pueden ser implementados como cualquier componente de React.

Los componentes que aquí se desarrollan están estilizados en base a gusto personal pero siguiendo prácticas de diseño de interfaces y experiencia de usuario, dando la posibilidad de que cuando se implemente algún elemento a este se le puedan dar parámetros básicos como podrían ser el color o el tamaño.

También cada uno de los componentes están conectados con el estado de React permitiendo un veloz desarrollo de la página Web que se desea hacer con esta biblioteca, esto significa que al realizar el maquetado automáticamente ya se tendrá un control de los datos.

Al finalizar el desarrollo se tendrá una biblioteca probada y publicada en el gestor de paquetes NPM, para que cualquier persona pueda acceder a esta.

# 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 [20] 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 [23] 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 marcos de trabajo, 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 biblioteca número uno es Node.js. El desarrollo Front-End [6] 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 bibliotecas o marco de trabajos, 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 [25], 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 desarollo, probó y decidió compartirlo, haciendo que el trabajo sea más ágil. De igual manera nosotros podemos aportar publicando nuestra biblioteca. Dentro de NPM contamos con miles de bibliotecas de código abierto. Teniendo esto en cuenta, nos da la posibilidad de tener nuestra propia biblioteca 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 [24] 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 marco de trabajo 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 [2]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 biblioteca 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 biblioteca 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 cuales serán elementos definidos, que contarán con una definición de estilos [3] (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 biblioteca. Otra ventaja por parte de la biblioteca es que esta estará basada en prácticas modernas del diseño UI/UX, como lo es Mobile First [14] 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. Esto nos dará la garantía de la experiencia del usuario, lo que significa que el usuario sabrá que está haciendo en todo momento, y también estaremos otorgando un diseño de interfaz moderno. Ya que el diseño de cada elemento de la biblioteca estará diseñado para ser fácil de usar para el desarrollador y usuario final y garantizar que el resultado del desarrollador sea el mejor para el usuario final. Finalmente, además de los elementos básicos que incluirá la biblioteca, 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 biblioteca 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 marco de trabajos 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 biblioteca.

Con el uso de esta biblioteca 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

biblioteca nos aseguraremos de que las vistas de nuestra aplicación Web puedan mirarse estéticamente iguales y tendemos 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 biblioteca evitamos el tiempo de aprender un nuevo marco de trabajo, ya que funciona sobre React y los conocimientos necesarios son saber React. Tenríamos 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 biblioteca estandariza el trabajo. Las bibliotecas 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 marco de trabajo y biblioteca 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 [16]. Otra cosa por tomar en cuenta es que no se necesita aprender una sintaxis específica para usar esta biblioteca, pero es necesario conocer la sintaxis que se usa con React, ya que esta biblioteca 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 biblioteca 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 biblioteca en un proyecto que ya está avanzado, y no lo es por el funcionamiento o incompatibilidad de bibliotecas, 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 biblioteca debemos contar el peso de nuestra biblioteca en sí, aunque no todas las tecnologías mencionadas son agregadas en la versión que estará alojado en el proyecto, ya que por ejemplo Webpack [8], solo es usado durante el desarrollo.

Al usar la presente biblioteca se acorta el trabajo necesario, que es requerido después de que el maquetado de nuestra página Web está listo, ya que a diferencia de otras bibliotecas como Material Design solo se enfocan el maquetado pero esta biblioteca se involucra en la funcionalidad de los componentes.

Una vez un componente de esta biblioteca, se implemente, este no necesitará trabajo extra, el componente estará listo para ser utilizado, por ejemplo para el componente de entrada de texto estará directamente conectado con el estado de React, eso significa que cuando necesitemos el dato introducido ya lo tendremos listo, y no como en React simple que debemos procesar el input para poder obtener el dato.

Esto deja la biblioteca como una opción híbrida entre el maquetado simple conformado por HTML, CSS etc. y el procesamiento de los datos con JavaScript y PHP.

# **Capítulo 3**

## **Objetivos**

### **3.1. Objetivos Generales**

Desarrollar una biblioteca para el desarrollo de páginas Web, especialmente para el área de front-end, publicada en el gestor de paquetes de JavaScript “NPM”, usando el marco de trabajo React, permitiéndonos la utilización de componentes para facilitar, agilizar y mejorar el maquetado de una página Web, incluyendo prácticas de diseño UI/UX que permiten que el usuario llegue a entender el funcionamiento de la aplicación sin que se requiera un entrenamiento de la misma, haciendo el sitio que implemente la biblioteca fácil de usar y atractiva visualmente.

### **3.2. Objetivos Específicos**

- Desarrollar una biblioteca para el desarrollo Front-End de páginas Web.
- Publicar la biblioteca para su uso con el gestor de paquetes de JavaScript [12] llamado NPM.
- Usar el marco de trabajo React que nos permite la creación de componentes reutilizables.
- Implementar Web-pack para garantizar el funcionamiento de la biblioteca ya que nos permite empaquetar y exportar todos los módulos y dependencias que incluye nuestra

bibliotecabiblioteca en un solo archivo para la correcta y ágil implementación.

- Aunado a esto se utilizará Babel [26] 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 bibliotecabiblioteca, 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

bibliotecaEl 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 biblioteca React, ya que nuestro trabajo funciona sobre React.

Esta biblioteca se encuentra publicada en el gestor de paquetes de JavaScript llamado NPM.

El rol que toma más partido en el uso de nuestra biblioteca es el Front-End, ya que es el área que desarrolla la parte visual para el software. La biblioteca 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 biblioteca 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 biblioteca 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

## Alcances 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 biblioteca 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 marco de trabajo Jest así garantizando la funcionalidad esperada para los usuarios que desean usar la biblioteca, 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

- **biblioteca Front-end:** Una biblioteca Front-end [22]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 [25] 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 [19] es una biblioteca 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 marco de trabajo 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 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. Los tres elementos principales de React son los siguientes:

- **Componente:** Una manera fácil de explicar un componente en React es compararlo con la programación orientada a objetos, en la cual un componente es un objeto, en el que este es una base ya establecida con propiedades. Si se instancian varios objetos de el mismo tipo, estos tendrán características en común pero con ciertas propiedades que los hacen distintos entre ellos. Con este principio podemos crear varios componentes en los cuales solo cambia algún elemento diferencial, como un texto una imagen etc.
- **Estado:** El estado en react siguiendo la analogía de la programación orientada a objetos, son los atributos de la clase, pero estos son internamente manejados por React, esto se refiere a que para actualizar el estado se debe usar una función proporcionada por la biblioteca.
- **Props:** Los props son los parámetros que se le da a nuestro componente para que este actúe de manera diferente a los otros componentes de el mismo tipo.

React crea un árbol en el cual hace una copia de el DOM que guarda en la memoria RAM, una de las principales ventajas de React es la que aquí se trata. Una vez el árbol creado, este espera a que alguno de los miembros del estado se actualice y actualiza el nodo que depende de ese estado. Con esto actualiza el DOM del navegador pero solo la parte diferencia, agilizando todo el proceso.

Por ejemplo si tenemos una lista nombres en nuestra Web y el usuario borra uno, el navegador quitará el elemento que el usuario elige, pero no se volverá a hacer el

render de toda la Web.

Existen dos componentes dentro de React uno es el componente clase y otro el componente función, con los que se puede tener los mismos resultados pero se usan a criterio propio.

En la siguiente imagen se encuentra un componente tipo clase, en el cual nos sirve para hacer una calculadora que suma o resta un parámetro dado.



```
1 class Calculadora extends React.Component {
2
3     constructor(props) {
4         super(props);
5         this.state = {
6             total: 0,
7         };
8         this.sumar = this.sumar.bind(this);
9         this.restar = this.restar.bind(this);
10    }
11
12    sumar() {
13        this.setState({
14            total: this.state.total + this.props.diferencia
15        });
16    }
17
18    restar() {
19        this.setState({
20            total: this.state.total - this.props.diferencia
21        });
22    }
23
24    render() {
25        return (
26            <>
27                <a onClick={this.restar}> - </a>
28                {this.state.total}
29                <a onClick={this.sumar}> + </a>
30            </>
31        );
32    }
33 }
```

Figura 7.1: Componente calculadora

En la línea 5 dentro del constructor se inicializa el estado de nuestro componente el cual guarda el total acumulado.

Dentro del return que está en el render tenemos dos etiquetas `<a>`( Línea 27 y 29 ) las cuales son los botones de suma y resta de nuestra aplicación, estas hacen un

llamado a las funciones de suma y resta dependiendo el operador. En la línea 28 mostramos el total acumulado obteniéndolo de el estado mediante `this.state` más el nombre del elemento del estado.

Las funciones sumar y restar actualizan el estado mediante `this.setState`, función que recibe un objeto con los datos de JavaScript que va a actualizar. En este caso suma o resta un parámetro dado por `props`.

La forma de hacer el llamado a un componente es la siguiente.



Figura 7.2: Llamado a componente

- **HTML** : HTML [5] 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 [3] 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 [8] 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 [26] 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.

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

Figura 7.3: Impresión de texto con React

El código será convertido en el que se muestra En la figura 7.4, el cual es interpretada por un mayor número de navegadores.

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

Figura 7.4: 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 biblioteca 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 biblioteca

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.

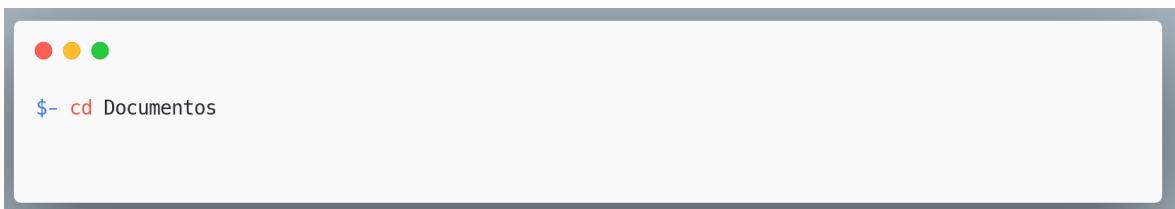
```
● ○ ●  
updateState(event) {  
  this.setState({  
    [event.stateName]: event.value  
  });  
}
```

Figura 8.3: Actualización del estado

## 8.1. Creación Del Ambiente De Desarrollo

### 8.1.1. Ambiente De Desarrollo

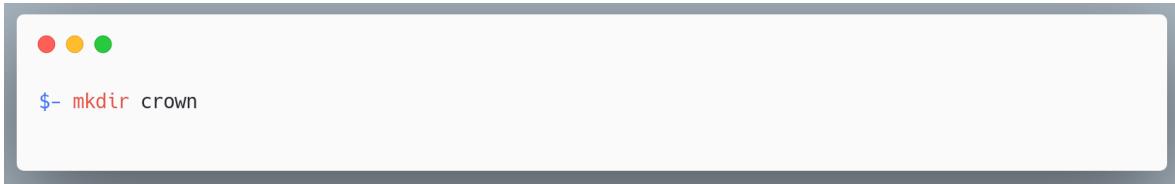
Durante el desarrollo de la presente biblioteca se usó el sistema operativo MacOS, con la terminal que se incorpora por defecto en el mismo. El primer paso que se debe realizarse es colocarse en el directorio en el que se desea trabajar, esta biblioteca se sitúa en la carpeta Documentos durante el desarrollo, usando el comando de la Figura 8.4 es posible cambiar de directorio, ejecutandolo en la terminal.



```
● ● ●
$- cd Documentos
```

Figura 8.4: Moverse entre directorios

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



```
● ● ●
$- mkdir crown
```

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.



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

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.



```
$- npm init
```

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 biblioteca.
- **Descripción del paquete:** Daremos a las personas una muy breve explicación acerca del uso que puedes obtener con nuestra biblioteca.
- **Punto de entrada del paquete:** Es el directorio el cual será importado cuando agreguemos nuestra biblioteca 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 biblioteca, 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

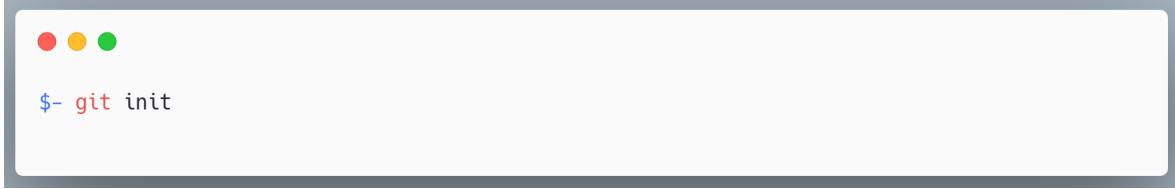
```

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”).



```
$ git init
```

A screenshot of a Mac OS X terminal window. The window has a dark grey header bar with three colored window control buttons (red, yellow, green) in the top-left corner. The main body of the window is white and contains a single line of text in a monospaced font: '\$ git init'.

Figura 8.9: Inicializar GIT

Creará 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"  
}
```

A screenshot of a Mac OS X terminal window. The window has a dark grey header bar with three colored window control buttons (red, yellow, green) in the top-left corner. The main body of the window is white and contains a JSON object in a monospaced font. The object has a single key, 'repository', which is itself an object with two properties: 'type' set to 'git', and 'url' set to 'git+https://github.com/FerCorona/crown.git'. There is a closing brace '}' at the end of the object.

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 biblioteca 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.



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

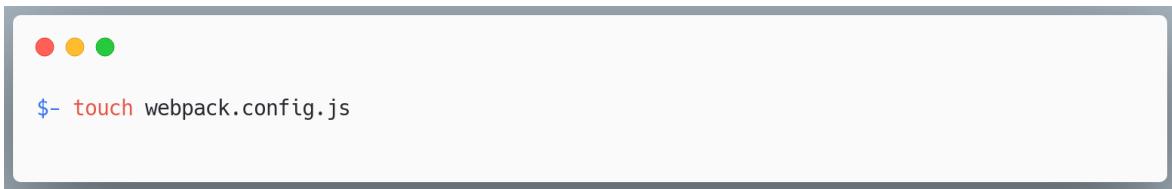
Figura 8.12: Agregar scripts

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.

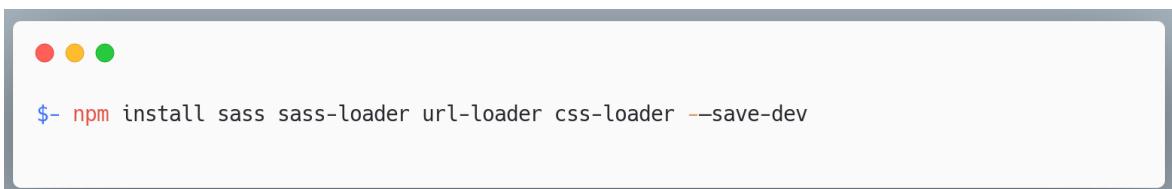
Para agregar configuración es necesario crear un archivo como se ilustra con el siguiente comando, estando en el directorio raíz de “crown”.



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

Figura 8.13: Crear archivo vacío

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



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

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: /\.s[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

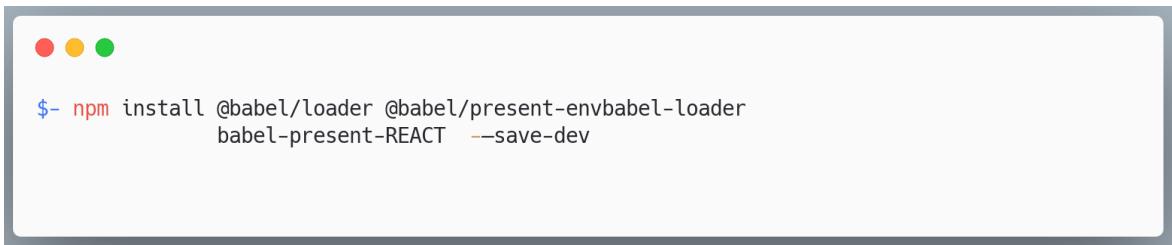
La anterior configuración muestra cómo debe procesar cada tipo de archivo que encuentra dentro de nuestro proyecto, por eso dada una expresión regular que define extensiones de archivos puede usar un cargador a usar. Lo que es definido en la configuración es:

- **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 biblioteca 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/present-env babel-loader  
babel-present-REACT --save-dev
```

Figura 8.16: Archivo Babel

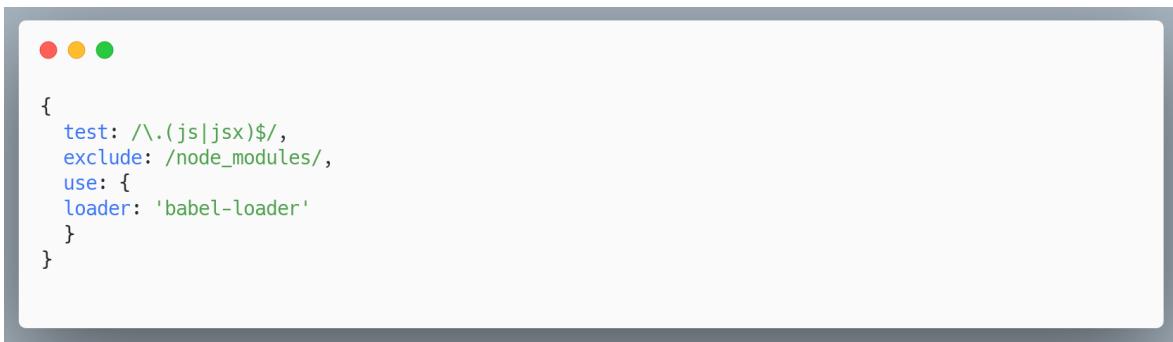
Después debemos crear un archivo en el directorio raíz llamado “.Babelrc”, al cual debemos agregar la siguiente configuración.



```
{  
  "presets": [ "@babel/present-env", "@babel/present-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”.



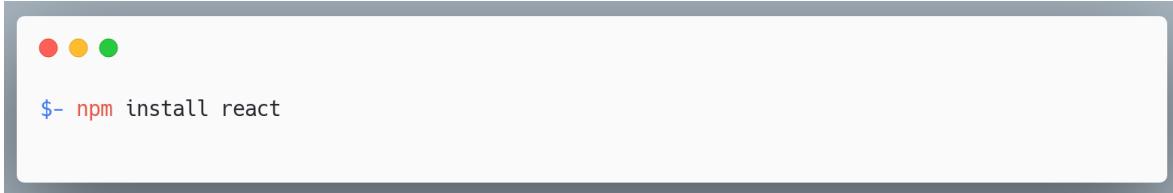
```
{  
  test: /\.(js|jsx)$/,  
  exclude: /node_modules/,  
  use: [  
    {  
      loader: 'babel-loader'  
    }  
  ]  
}
```

Figura 8.18: Agregar Babel en Webpack

Esto para que WEPACK 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 biblioteca y para agregarlo es necesario ejecutar el siguiente comando.



```
$- npm install react
```

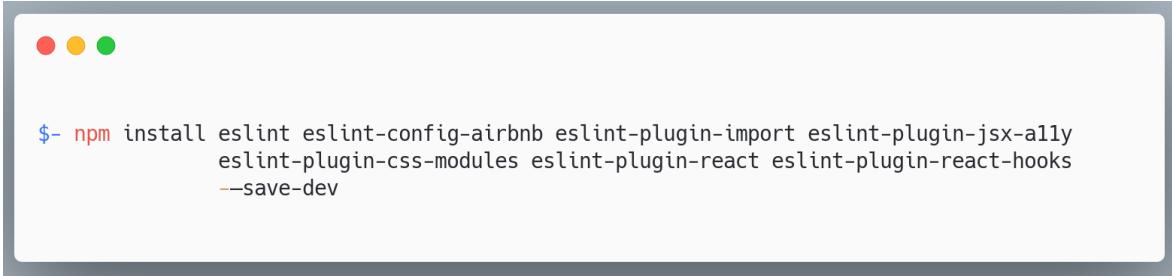
Figura 8.19: Agregar React

Para este comando no se agrega la bandera “–save-dev” al final, por que de esta manera forzamos a que cuando se instale esta biblioteca también se instale React en caso de que no estuviera, ya que nuestra biblioteca 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

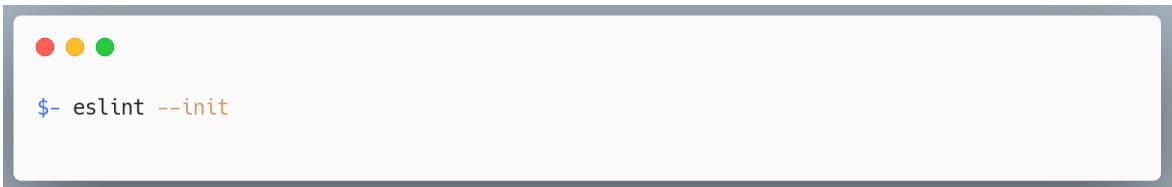
que en toda la biblioteca tengamos un código unificado. De igual manera no tendremos que preocuparnos por esto si no que al guardar el archivo obtenga el formato correcto. Para esto debemos ejecutar las siguientes dependencias en la terminal.



```
$- 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
```

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.



```
$- eslint --init
```

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.
- **Marco de trabajo 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 biblioteca.

## 8.2. Lineamientos UX/UI

Uno de los puntos clave para obtener los resultados esperados de la biblioteca, es tomar en cuenta la experiencia de usuario ( UX ) y la interfaz de usuario ( UI ), que podrían definirse de la siguiente manera.

- **Experiencia de usuario ( UX ):** Es la sensación que una persona percibe, al usar una herramienta como puede ser una página web, o aplicación móvil, esto incluye la funcionalidad y la respuesta que recibe, logrando que una persona sienta que ya conoce la aplicación y conocer cada acción que realiza.
- **Interfaz de usuario ( UI ):** Es la percepción visual de una web o aplicación móvil, esta puede ser agradable para el usuario y generar el interés o rechazo de la misma, y llegar a sentir emociones.

Se han encontrado una serie de parámetros, los cuales se logran obtener una interfaz y experiencia de usuario favorables, que serán implementados en la presente biblioteca.

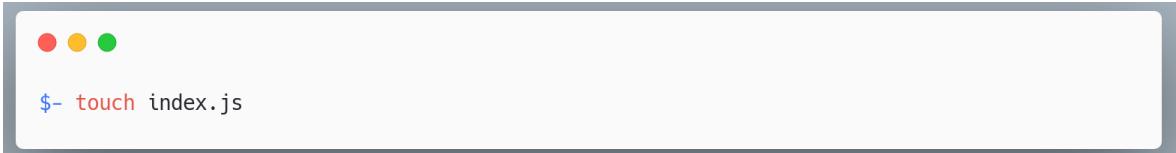
- **Sombras:** El uso de las sombras en los elementos de una interfaz gráfica, genera un mayor interés entre los elementos mostrados y da el efecto de tener una mayor altura dimensional.
- **Jerarquía de elementos:** Para que el usuario entienda el objetivo de la aplicación Web debe existir una jerarquía visual, para que el usuario entienda cual es el objetivo de la Web representado por elementos visualmente más grandes y los elementos por los cuales no tendrían que tener tanta importancia serán más pequeños.
- **Alineación vertical y horizontal:** Este punto puede ser visto como generar una cuadrícula, la cual cada elemento coincide vertical y horizontalmente.
- **Retroalimentación:** Para que el usuario tenga la sensación de que tiene todo el control de la Web y que sienta que sabe que está haciendo siempre, se deben proporcionar efectos para que esto sea posible, por ejemplo cuando pasas el mouse sobre algún elemento este debe reaccionar.

- **Tipo de fuente:** Para la correcta legibilidad de cualquier texto en la biblioteca se usará la fuente llamada San Francisco desarrollada por Apple de la familia de las fuentes Sans-Serif.
- **Alto de línea:** Si se aumenta el alto de la línea permitirá que en un texto sea más legible porque existirá más espacio y no se tendrá texto que parezca encimado.
- **Consistencia:** Este punto trata de que todos los elementos tengan similitudes entre sí, no importa si sean del mismo tipo, esto será fácil para que el usuario conozca cada una de las pantallas aunque sea la primera vez que se visita.
- **Colores:** Existe una actual tendencia por el uso de colores pasteles, ya que estos te hacen sentir familiar, ya que estos están presentes mayormente en la naturaleza.

## 8.3. Desarrollo

### 8.3.1. Configuración Archivo Inicial

El proyecto contendrá un nodo raíz el cual será el que tendrá los llamados al conjunto de elementos que tendrá nuestra biblioteca. 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.



```
$- touch index.js
```

Figura 8.22: Inicializar archivo vacío

Dentro de este archivo debemos poner el llamado a cada elemento que se vaya agregando a nuestra biblioteca ( 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';

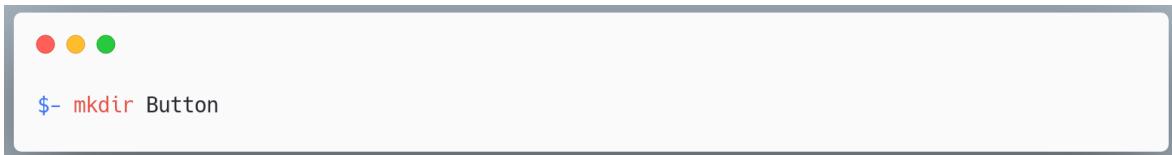
export {
  Button,
  ElementoN
};
```

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.

### 8.3.2. Elemento Botón

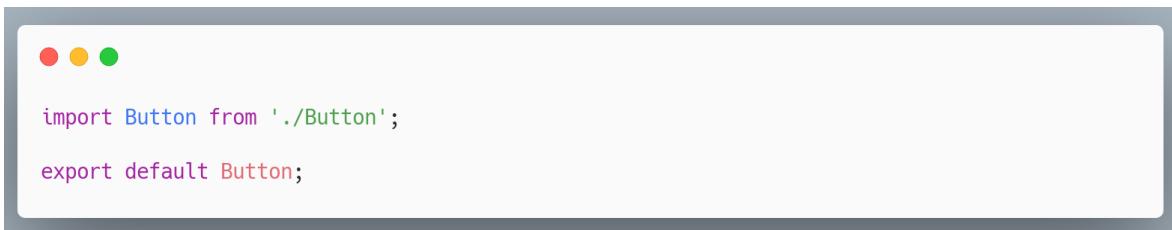
El primer elemento que vamos a agregar es el botón, para esto creamos un directorio llamado “Botton” de la siguiente manera.



```
$- mkdir Button
```

Figura 8.24: Crear directorio para el elemento Button

Y dentro de este directorio crearemos dos archivo 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.



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

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 agregarán una curvatura, tenemos Round y SemiRound uno más 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**

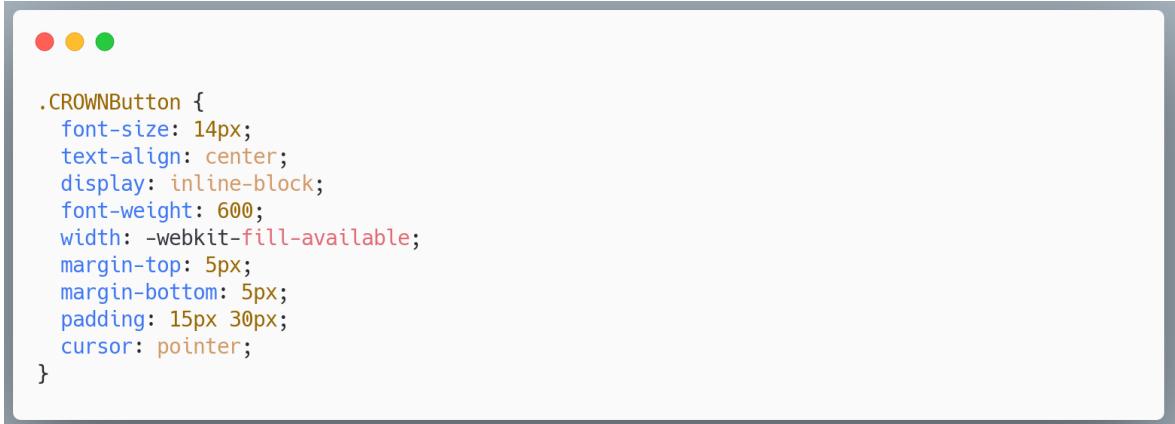
Los colores que se usan son una constante definida por la biblioteca ya que se considera que son cromáticamente compatibles entre ellos, este tema se abordará más adelante. A continuación se muestra un fragmento del código que permite agregar la configuración requerida.



```
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

### 8.3.3. 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

### 8.3.4. Elemento Input Tex

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.l
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 Input Text, 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='light_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

### 8.3.5. 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.	Función vacía.
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>
          )
        );
      }
    );
  }
}
```

Figura 8.30: Código fuente del elemento DropDown

### 8.3.6. 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.	Función vacía.
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



The screenshot shows a code editor window with a dark theme. At the top left, there are three circular icons: red, yellow, and green. The main area contains the following code:

```
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='ligth_x' />
        </div>
      </label>
    ))
  }
</div>
);
```

Figura 8.31: Código fuente del elemento RadioButton

### 8.3.7. 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

### 8.3.8. 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

### 8.3.9. 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='light_x' />
        </div>
      )
    }
  </div>
);
```

Figura 8.34: Código fuente del elemento CheckBox

### 8.3.10. 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 biblioteca 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

### 8.3.11. 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 biblioteca 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**

```
● ● ●

const Loader = ({ show }) => (
  show && (
    <div className='CROWNLoader'>
      <div /><div /><div /><div /><div /><div /><div /><div /><div />
    <div /><div />
      </div>
    )
  );
);
```

Figura 8.36: Código fuente del elemento Image

### 8.3.12. 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>
  </div>
);
```

Figura 8.37: Código fuente del elemento Card

### 8.3.13. 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

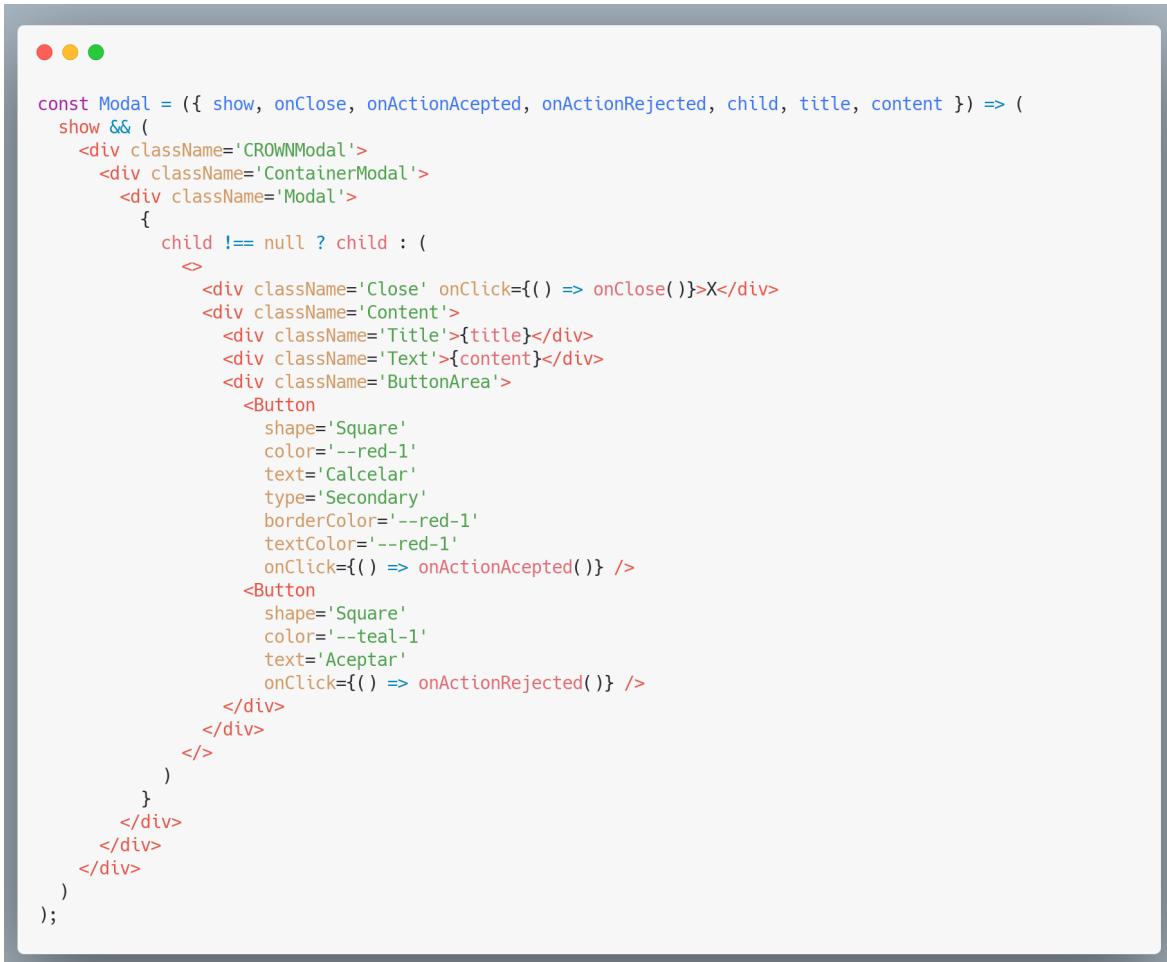
Los parámetros obligatorios para su funcionamiento son:

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

- **onActionAccepted**

- **onActionRejected**

- **content**



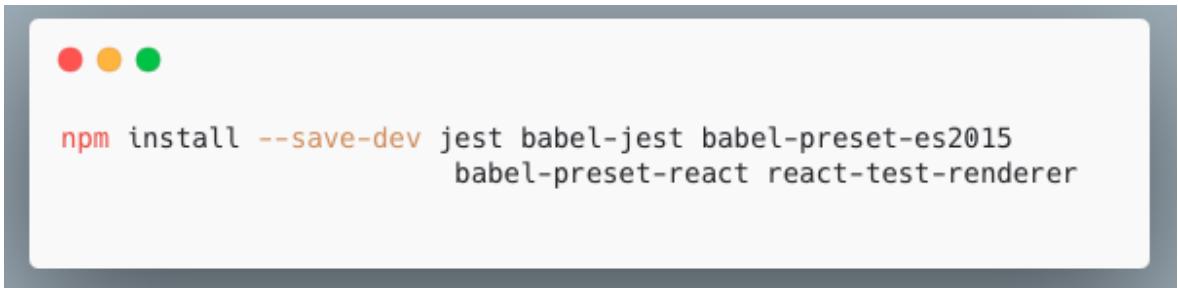
The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. The main area contains the following code:

```
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>
  );
);
```

Figura 8.38: Código fuente del elemento Modal

## 8.4. Test

Para garantizar el correcto funcionamiento de cada uno de los componentes se diseñan una serie de test para comprobar el comportamiento. Para esto se usará una librería llamada Jest [10] , para instalarla se debe ejecutar el siguiente comando.



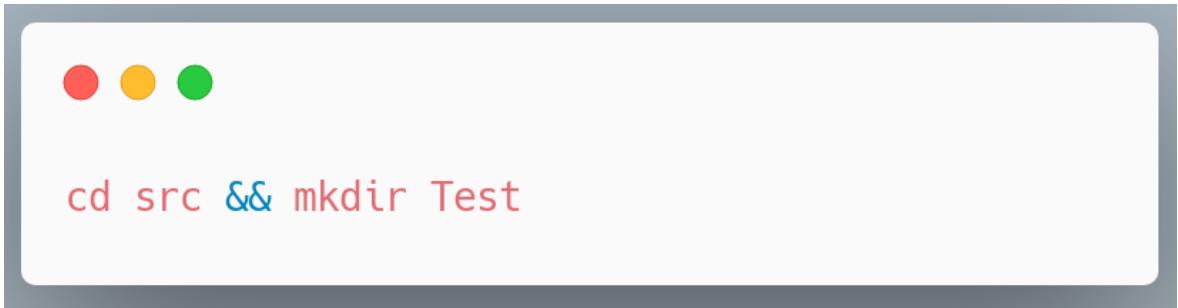
```
npm install --save-dev jest babel-jest babel-preset-es2015  
babel-preset-react react-test-renderer
```

A screenshot of a Mac OS X terminal window. The window has three colored title bar buttons (red, yellow, green) at the top left. The main area contains the command 'npm install --save-dev jest babel-jest babel-preset-es2015 babel-preset-react react-test-renderer' in black text on a white background.

Figura 8.39: Instalar dependencias

Las dependencias instaladas con el comando anterior solo son usadas durante el desarrollo.

Dentro del directorio src debemos crear un subdirectorio llamado Test.



```
cd src && mkdir Test
```

A screenshot of a Mac OS X terminal window. The window has three colored title bar buttons (red, yellow, green) at the top left. The main area contains the command 'cd src && mkdir Test' in black text on a white background.

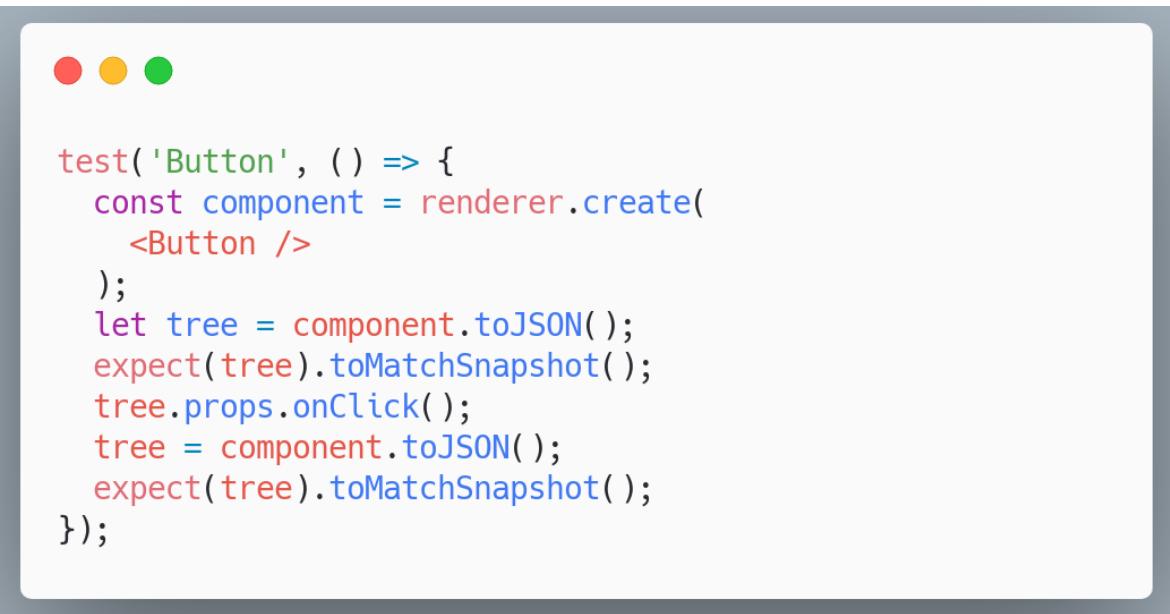
Figura 8.40: Crear directorio

Y dentro de este directorio crearemos un archivo por cada uno de los componentes llamado de la siguiente manera.

Nombre del Componente + . + test + . + js

Dentro de este se tendrán un código para asegurar el funcionamiento, por ejemplo para

el componente Button se diseñó el siguiente test.



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The main area contains a Jest test script for a 'Button' component. The code uses the renderer API to create a component, then checks its JSON representation before and after performing an onClick action. The file path 'biblioteca' is visible at the bottom left of the terminal window.

```
test('Button', () => {
  const component = renderer.create(
    <Button />
  );
  let tree = component.toJSON();
  expect(tree).toMatchSnapshot();
  tree.props.onClick();
  tree = component.toJSON();
  expect(tree).toMatchSnapshot();
});
```

Figura 8.41: Test del componente Button

El cual crea un componente, después crea una copia del resultado, y cada que se ejecute el test se debe comparar con la copia creada. La segunda parte del test hace click en el botón y verifica el funcionamiento.

Para poder ejecutar todos los test, se debe agregar el siguiente comando en el archivo package.json.



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The main area contains a command in green text: '"test": "jest src/Test --watchAll"'. This command is used to run all Jest tests in the 'src/Test' directory.

```
"test": "jest src/Test --watchAll"
```

Figura 8.42: Agregar comando

## 8.5. Publicación de la biblioteca en NPM

Finalmente una vez tenemos todos los elementos completamente desarrollados y terminados debemos proceder a publicar nuestra biblioteca para que esta sea de uso libre para toda la comunidad de desarrolladores de software Web que implementan Node.

Debemos configurar el nombre de nuestro paquete que será con el que las personas pueden encontrarlo, esta biblioteca se llama "crown components", este nombre se especifica en el archivo package.json en el directorio raíz de nuestro proyecto en la llave "name".

Tenemos que ingresar con nuestra cuenta desde la terminal con el siguiente comando.

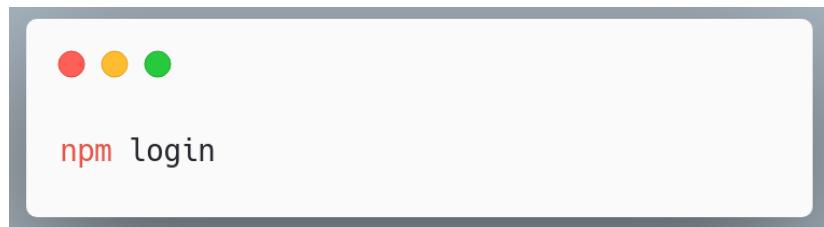


Figura 8.43: Ingresar a nuestra cuenta

Este comando solicitará el nombre de usuario, contraseña y correo electrónico de nuestra cuenta de NPM, es necesario tener una cuenta previamente creada.

Una vez con la sesión iniciada en la terminal es posible publicar nuestro paquete con el siguiente comando.

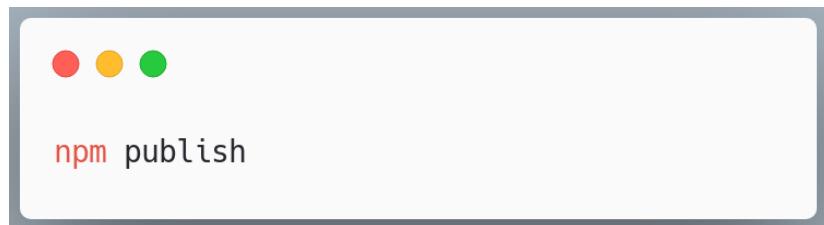


Figura 8.44: Publicar paquete

Y con esto tenemos nuestra biblioteca publicada.

## 8.6. Diferencia entre el uso nativo de React y esta biblioteca

Hasta este punto se tiene la biblioteca publicada en el gestor de paquetes NPM, pero aún no se tiene conciencia sobre la diferencia que existe entre usar esta biblioteca o implementar React de manera normal. A continuación se tiene un ejemplo de cómo se implementaría el Front-End para mostrar la lista de calificaciones de un grupo de estudiantes con React.

Queremos tener una vista que tenga el título “Lista de calificaciones” con una tabla que tenga el nombre, el semestre y tres calificaciones parciales.

Supongamos que al hacer una petición al servidor Back-end ( Esta biblioteca no cubre la manera directa de interactuar con el Back-end ), y nos regresa como respuesta un array con los datos y calificaciones de los alumnos como se muestra en la siguiente imagen, el array llamado ALUMNOS. También tenemos un array llamado HEADER el cual tiene la cabecera de la tabla.

```
● ● ●

const ALUMNOS = [
  ['Alumno 1', '1', '90', '67', '98'],
  ['Alumno 2', '1', '70', '70', '78'],
  ['Alumno 3', '1', '78', '67', '49'],
];
const HEADER = [ 'Nombre', 'Semestre', 'Parcial 1', 'Parcial 2', 'Parcial 3'];
```

Figura 8.45: Array de datos

En React tendríamos el siguiente código para poder mostrar en pantalla una tabla.



```
1 <div className='container'>
2   <p>Lista de calificaciones</p>
3   <table>
4     <thead>
5       <tr>
6         {
7           HEADER.map(titulo => (
8             <th>{titulo}</th>
9           ))
10        }
11      </tr>
12    </thead>
13    {
14      ALUMNOS.map(fila => (
15        <tr>
16        {
17          fila.map(celda => (
18            <td>
19              {celda}
20            </td>
21          ))
22        }
23      </tr>
24    ))
25  }
26 </table>
27 </div>
```

Figura 8.46: Array de datos

En la línea dos, tenemos el título de la vista, a partir de la línea tres tenemos la tabla, la primera parte ( línea cuatro) hacemos la cabecera de la tabla, para esto tenemos que recorrer el array llamado HEADER con ayuda de la función de JavaScript llamada map.

Después en la línea trece, formamos el contenido de la tabla, tenemos que recorrer el array ALUMNOS, primero se recorre fila por fila, y en la línea diecisiete dada una fila, se recorre celda por celda.

Para que esto luzca de manera deseada se tiene el siguiente CSS.

```
1 .container {  
2   padding: 10%;  
3 }  
4 .formContent {  
5   margin: 30px 0;  
6 }  
7 p {  
8   margin-top: 10px;  
9   margin-right: 0px;  
10  margin-bottom: 5px;  
11  margin-left: 0px;  
12  font-size: 22px;  
13  font-weight: 600;  
14  color: #2F2F2F;  
15 }  
16 table {  
17   margin-top: 30px;  
18   border-collapse: collapse;  
19   box-shadow: 0 0 8px 0 #d3d3d3;  
20   border-radius: 10px;  
21   overflow: hidden;  
22   margin-bottom: 5px;  
23   width: 100%;  
24 }  
25 thead {  
26   color: white;  
27   background-color: #ADDCE5;  
28   height: 50px;  
29   color: white !important;  
30 }  
31 tr {  
32   text-align: center;  
33   height: 40px;  
34 }
```

Figura 8.47: Array de datos

Y así tenemos el siguiente resultado.

Lista de calificaciones				
Nombre	Semestre	Parcial 1	Parcial 2	Parcial 3
Alumno 1	1	90	67	98
Alumno 2	1	70	70	78
Alumno 3	1	78	67	49

Figura 8.48: Array de datos

Con nuestra biblioteca evitamos el trabajo de formar desde cero la tabla, y no se tiene que dar estilos CSS, Para tener el resultado anterior no tenemos que hacer nada, más que pasarle los datos de la cabecera y el cuerpo de la tabla de la siguiente manera.



```

● ● ●

<div className='container'>
  <Label
    text='Lista de calificaciones'
    color='--black-2'
    weight='600'
    size='22px' />
  <div className='formContent'>
    <Table
      body={ALUMNOS}
      header={HEADER}>
    />
  </div>
</div>

```

Figura 8.49: Array de datos

Con el código anterior, estamos usando la biblioteca aquí desarrollada, al inicio tenemos el componente Label al cual le damos parámetros como el tamaño de la letra, su peso

y el texto a mostrar que en este caso es “Lista de calificaciones”. Y después tenemos el componente Table, al cual solo le pasamos los arrays ALUMNOS para el cuerpo de la tabla y HEADER para la cabecera de la tabla.

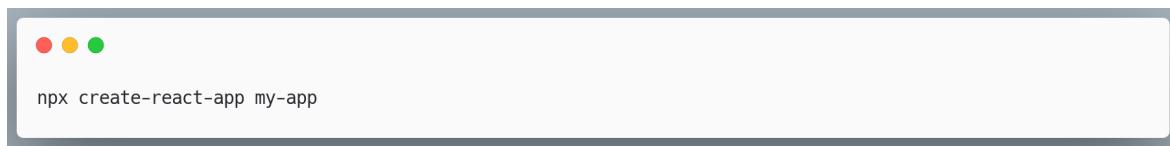
# Capítulo 9

## Pruebas y Resultados

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

Existe una herramienta que nos permite crear el esqueleto de una Web basada en React. [9] esta es desarrollada y mantenida por Facebook, de la cual partiremos para crear una Web en la que probaremos la biblioteca.

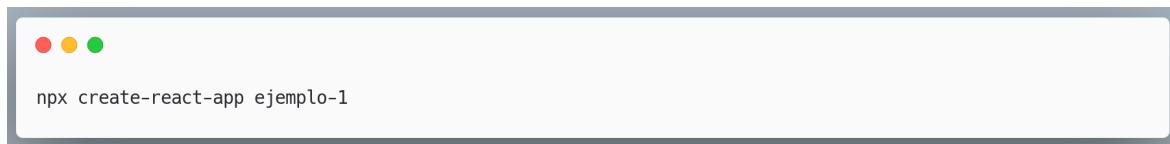
Instalaremos la herramienta llamada Create React App con el siguiente comando.



```
npx create-react-app my-app
```

Figura 9.1: Instalar Create React App

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



```
npx create-react-app ejemplo-1
```

Figura 9.2: Crear una nueva app

El comando nos generará los siguientes directorios.

```
> node_modules
└── public
    ├── favicon.ico
    ├── index.html
    ├── logo192.png
    ├── logo512.png
    └── manifest.json
    └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── reportWebVitals.js
    ├── setupTests.js
    └── .gitignore
    └── package-lock.json
    └── package.json
    └── README.md
```

Figura 9.3: Directorios al crear una nueva app

Para poder probar nuestra biblioteca debemos indicar a la app ejemplo-1 que use la dependencia instalándola con el siguiente comando.

```
● ● ●
npm i crown_components
```

Figura 9.4: Instalando nuestra biblioteca

Ahora basta incluirlo como se agrega cualquier dependencia de node, basta con agregar cualquiera de los elementos aquí desarrollados.

Crearemos dos ejemplos simples para mostrar el funcionamiento de la biblioteca desde el lado Front-end, cabe aclarar que no se tratará el tema de cómo procesar los datos hacia el Back-end. El primero consta de un formulario el cual solicita nombre, correo electrónico y contraseña, esto con el objetivo de registrar un usuario para que tenga un correo institucional, también se le solicitará si la cuenta estará activa al momento de crearla.

Para esto debemos importar nuestros componentes de la siguiente manera.

```
import { Label, InputText, Button, Table, Switch } from 'crown_components';
```

Figura 9.5: Implementación la biblioteca

Después en el estado de React debemos agregar algunos campos para almacenar la información del formulario como se muestra en la siguiente figura.

```
this.state = {
  nUsuario: '',
  correo: '',
  password: '',
  isActive: false,
  users: []
};
```

Figura 9.6: Agregar campos al estado de React

Los datos se usarán para:

- **nUsuario:** Nombre de usuario del nuevo registro.
- **correo:** Correo del usuario.
- **password:** Contraseña del usuario.
- **isActive:** Activar la cuenta al crearla.
- **users:** Usuarios registrados, esto simulando nuestra base de datos.

En la siguiente imagen se muestra el código necesario para desplegar en pantalla el formulario requerido así como una tabla para mostrar los usuarios registrados.

The screenshot shows a code editor window with a dark theme. At the top, there are three circular icons: red, yellow, and green. The main content area contains the following React component code:

```
<Label
  text='Registrar nuevo usuario'
  color='--black-2'
  weight='600'
  size='22px' />
<div className='formContent'>
  <InputText
    placeholder='Nombre de usuario'
    value={this.state.nUsuario}
    stateName={'nUsuario'}
    onChange={this._updateState}>
  </InputText>
  <InputText
    placeholder='Correo electronico'
    value={this.state.correo}
    stateName={'correo'}
    onChange={this._updateState}>
  </InputText>
  <InputText
    placeholder='Contraseña'
    value={this.state.password}
    stateName={'password'}
    onChange={this._updateState}
    type='password'>
  </InputText>
  <Switch
    text='Activar usuario'
    stateName='isActive'
    onChange={this._updateState}>
  </Switch>
  <Button
    onClick={this._addUser}
    text='Registrar'
    color='--green-3'
    borderColor='--green-3'
    shadow={false}>
  </Button>
</div>
<Label
  text='Usuarios registrados'
  color='--black-2' weight='600'
  size='16px' />
<Table
  body={this.state.users}
  header={[ 'Nombre', 'Correo', 'Activo' ]}>
```

Figura 9.7: Código para maquetar

El código contiene al inicio un componente Label el cual muestra en pantalla un texto

dado con las especificaciones de preferencia, se le da un tamaño de letra y color del texto. Después siguen una serie de `InputText` los cuales son la entrada de datos para el nombre de usuario, correo electrónico y contraseña. Estos tienen parámetros que son importantes de mencionar ya que son indispensables para el correcto funcionamiento como son: `placeholder`: Es el texto que se mostrará en el campo de texto antes de que se introduzca un dato, para dar una pista del dato solicitado. `value`: Es el estado de React en el que se almacenará cuando ocurra un cambio. `stateName`: Es la forma en como es llamado en el estado de React ( Ejemplo. en este caso sería alguna de las llaves que se muestran en la Figura 9.6 ). `onChange`: Es la función que se activará cuando algún dato se actualice.

El último parámetro `onChange` es una función simple como la que se muestra a continuación.



```
_updateState(update) {  
  this.setState({  
    [update.stateName]: update.value  
  })  
}
```

Figura 9.8: Función para actualizar el estado de React

Esta función es única para cualquiera de los componentes de la biblioteca, ya que están diseñados para que cuando se actualicen, regresen el nombre por el cual están identificados en el estado de React para saber qué dato actualizar y el nuevo valor. Finalmente se tiene el componente `Button` y `Table`. El componente `Button` tiene un parámetro, `onClick` el cual envía una función que se ejecutará cuando se haga click. Esta función actualiza el parámetro del estado llamado `users` que almacena de manera no persistente los usuarios que se vayan agregando. El componente `Tabla` recibe un arreglo para la cabecera y un arreglo de arreglos para el cuerpo de la tabla.

A continuación se muestra el resultado que se tendría con unas escasas líneas de código usando la biblioteca.

**Registrar nuevo usuario**

Nombre de usuario

Correo electronico

Contraseña

Activar usuario

**Registrar**

**Usuarios registrados**

Nombre	Correo	Activo
--------	--------	--------

Figura 9.9: Resultado final

También se tiene el resultado después de agregar algunos usuarios.

### Registrar nuevo usuario

usuario3

usuario3@app.com

.....

Activar usuario

**Registrar**

Usuarios registrados		
Nombre	Correo	Activo
usuario1	usuario1@app.com	No
usuario2	usuario2@app.com	Si
usuario3	usuario3@app.com	Si

Figura 9.10: Resultado final al agregar usuarios

Los datos que se agregaron en la tabla, solo están almacenados en la memoria volátil del navegador, ya que esta biblioteca no cubre el hacer la petición HTTP. Pero la biblioteca es capaz de usarse con cualquier otra biblioteca que se encargue de hacer las peticiones por ejemplo AXIOS [4].

En seguida se muestra otro ejemplo que figura un restaurante en el cual se muestran los platillos que se venden, con la posibilidad de filtrarlos si son bebidas o tragos.

En el estado de React se tiene solamente un dato, este es el modo por el cual se están filtrando los datos.



```
this.state = {
  filterBy: 'Comida'
};
```

Figura 9.11: Filtro en el estado de React

El código para maquetar los platillos es el siguiente el cual puede ser usado por ejemplo para la Web de un restaurante la cual recibe los datos de algún otro servicio Web. El código de la siguiente Web cuenta con elemento DropDown que permite filtrar para mostrar comida o tragos las opciones posibles para mostrar son enviadas por el parámetro options, stateName es el nombre de llave que almacena la opción por la que se filtra, y también se tiene optionSelected que es en si el filtro.

Después se tiene un ciclo en el que se agrega cada uno de los platillos, el componente Card recibe el título de la tarjeta, la imagen y el contenido.



```
<Label text='Menu' />
<DropDown
  stateName='filterBy'
  options={['Comida', 'Bebida']}
  optionSelected={this.state.filterBy}
  onChange={this._updateState} />
<div className='grid'>
{
  MENU.map(item => (
    item.type === this.state.filterBy ? (
      <Card
        title={item.title}
        src={item.src}
        content={item.content}>/<
      ) : null))
}
</div>
```

Figura 9.12: Código para mostrar los platillos

En las siguientes imágenes se muestra el resultado del código.

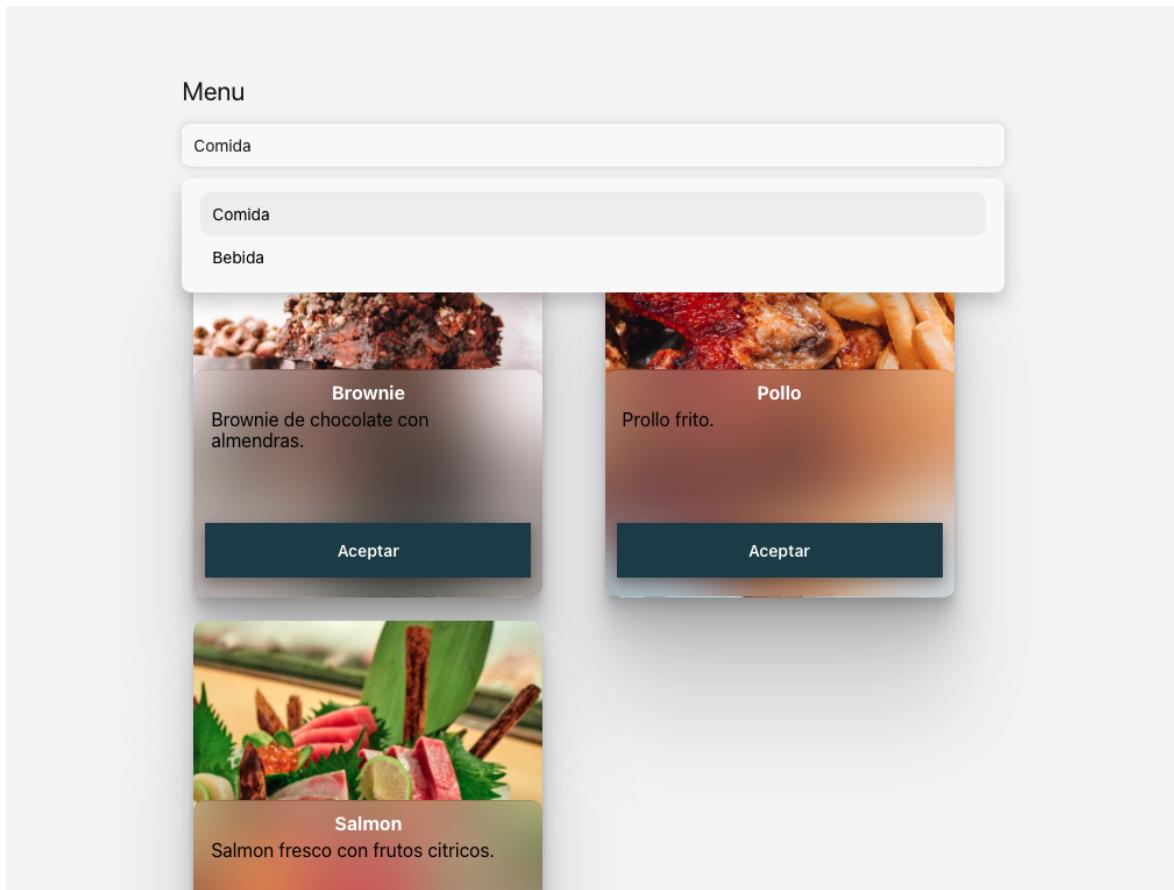


Figura 9.13: Filtrar por comida

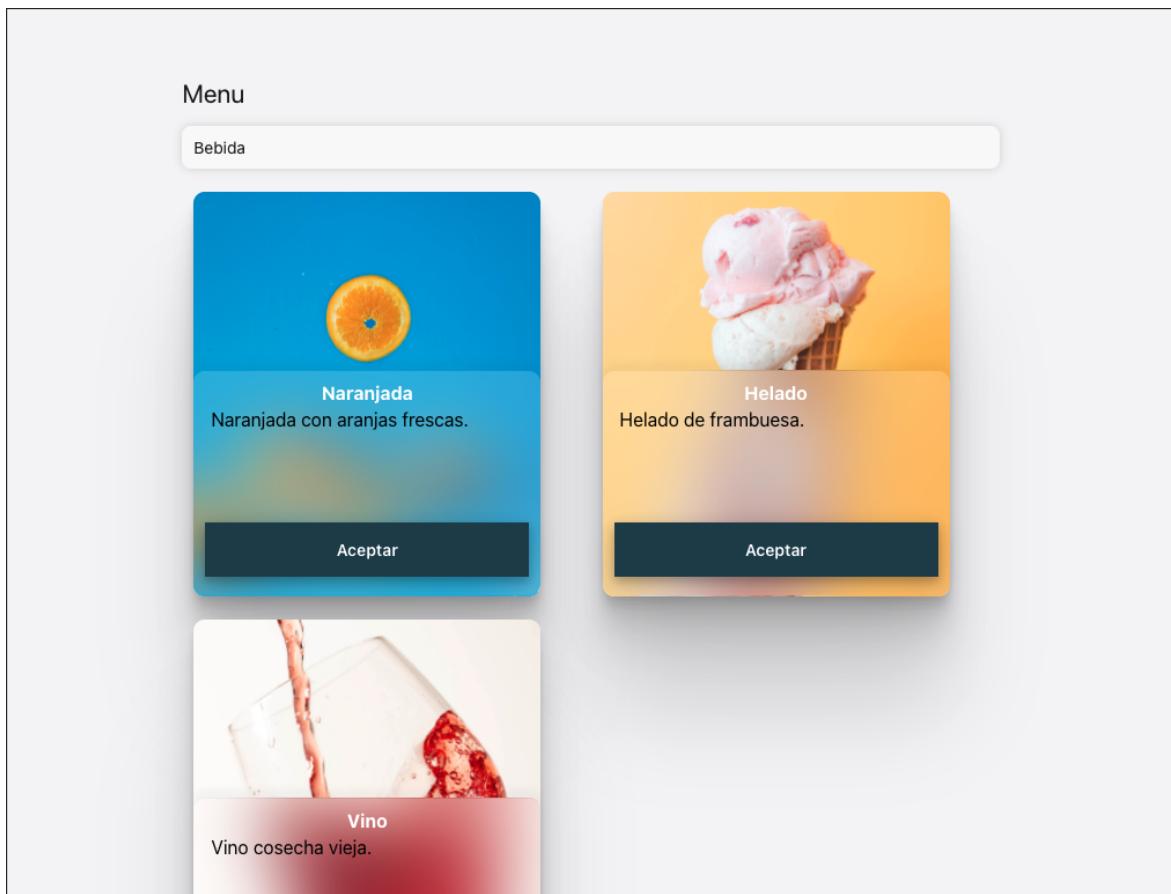


Figura 9.14: Filtrar por bebida

También se cuenta con el elemento Modal que puede ser usado como confirmación cuando el comprador presiona el botón para comprar un platillo.

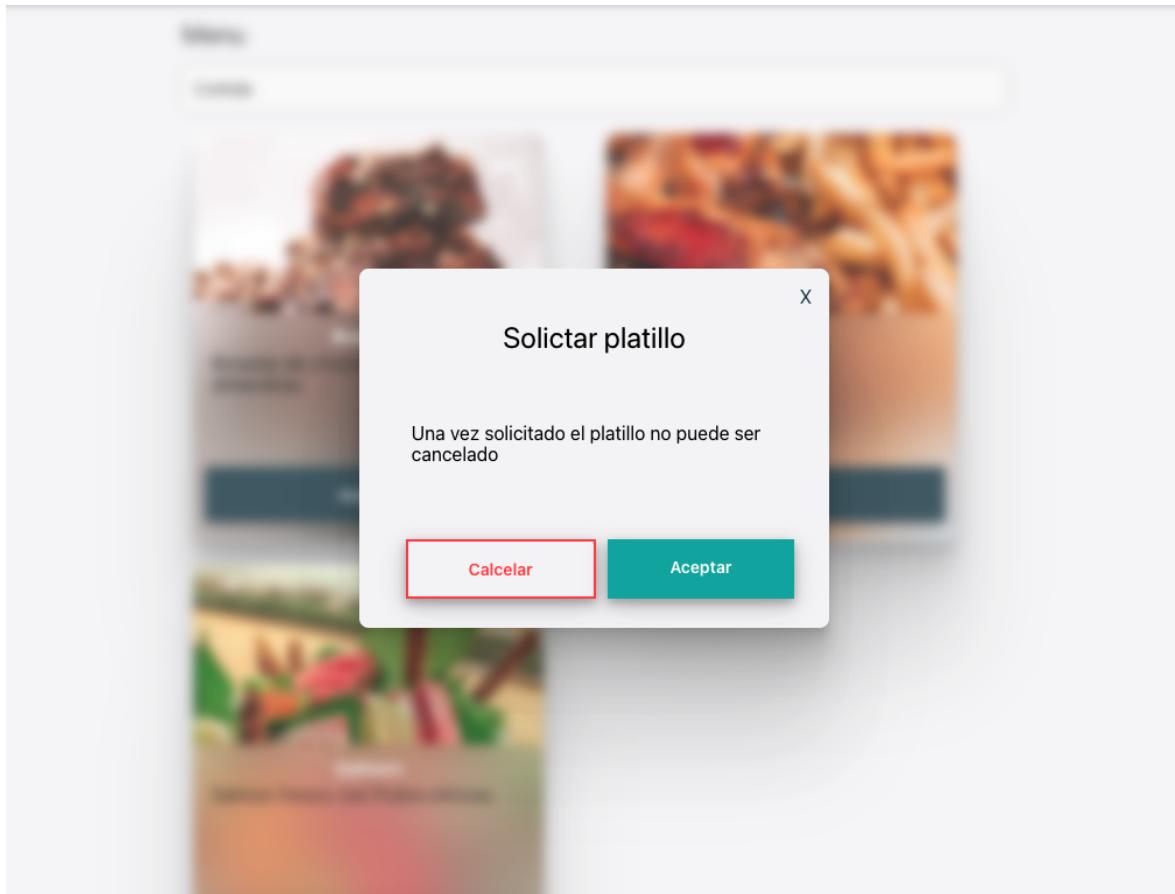


Figura 9.15: Confirmación de pedido

Los datos para mostrar los diferentes platillos no son obtenidos desde una petición HTTP, estos son tomados desde un array que existe sólo mientras la aplicación está en ejecución, ya que la biblioteca funciona para el Front-end. Se puede implementar cualquier tecnología para hacer peticiones y obtener datos.

## 9.1. Comparación contra Material Design

- **Número de componentes:** La biblioteca Material Design ya tiene algún tiempo de desarrollo, razón por la cual ya cuenta con gran cantidad de componentes, puedes encontrar múltiples opciones para cualquier necesidad, en contraste la biblioteca que aquí se desarrolla tiene un número limitado de componentes, que se considera que son los básicos usados en cualquier componente.
- **Facilidad de uso:** Como ya se mencionó, Material Design tiene variedad de com-

ponentes, pero también gran cantidad de opciones que se pueden modificar, lo cual presenta una desventaja ya que si un nuevo desarrollador comienza a utilizar Material Design tendrá una mayor curva de aprendizaje, con respecto a nuestra biblioteca.

- **Documentación:** Por el tiempo que tiene Material Design, ya tiene una gran cantidad de personas que lo saben usar, y foros en los que se habla sobre funcionamiento y dan consejos para un mejor uso.
- **Híbrido:** Material Design está diseñado para ser una biblioteca Front-end, pero nuestra biblioteca va más allá, la biblioteca que aquí se desarrolla para trabajar cien por ciento de la mano con el estado de react.
- **Peso:** Para comprobar el peso que requiere la biblioteca, se crean dos proyectos básicos de React [9] llamados proyectoCrown y proyectoMaterial, para el primero se instalará la biblioteca aquí desarrollada y en el segundo se instalará Material Design. Al momento de crear las carpetas cada una tiene un peso de 241 MB, esto por el peso de la carpeta conocida node modules que guarda todas las dependencias. Después se instaló cada una de las bibliotecas, el los proyectos. Una vez las bibliotecas instaladas el peso final de cada archivo fue el siguiente.
  - **projetoCrown:** 249 MB
  - **projetoMaterial:** 267 MB

Lo que nos da una diferencia de 18 MB de nuestra biblioteca con respecto a Material Design.

# Capítulo 10

## Conclusiones Y Recomendaciones

Una vez el desarrollo de la biblioteca finalizado, y la herramienta publicada, se recomienda usar como documentación las tablas de propiedades que se encuentran dentro del capítulo 8.2, tablas las cuales muestran las propiedades que pueden ser modificadas en cada uno de los componentes para su correcto funcionamiento. También se recomienda mantener siempre la versión más reciente que se encuentre publicada.

Finalmente se concluye con el pensamiento de no reinventar la rueda ya que existen múltiples bibliotecas que nos agiliza nuestro trabajo, y es fácil encontrar bibliotecas que se adecuen a cualquier proyecto en el que estamos trabajando. Así hacemos que lo complejo sea decidir una biblioteca y no el desarrollo en sí mismo.

# Capítulo 11

## Fuentes de Información

- [1] Agrawal, K. (2019, May 4). Airbnb Javascript style guide — Key takeaways - Docon. Medium. <https://medium.com/docon/airbnb-javascript-style-guide-key-takeaways-ffd0370c053>
- [2] Banks, A., Porcello, E. (2017). Learning REACT Functional Web Development with REACT and Redux. O'Reilly.
- [3] Coyier, C. (2013). SASS FOR WEB DESIGNERS. <https://abookapart.com/products/sass-for-web-designers>
- [4] Cruz, R. (2019, November 1). Using Axios For HTTP Requests - Rafael Cruz. Medium. <https://medium.com/@ralph1786/using-axios-for-http-requests-be9abb80795b>
- [5] Danny Goodman. (1998). Dynamic HTML The Definitive Reference. 101 Morris Street, Sebastopol, CA 95472: O'REILLY.
- [6] Fu, C. (2016). Exploration of Web front-end development technology and optimization direction.

- [7] 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>
- [8] Gallegos, G. (2017, 20 octubre). Para que es Webpack? Medium. <https://medium.com/tecninja/porque-usar-webpack-4a5004094455>
- [9] Getting Started | Create React App. (s. f.). Create React App. <https://create-react-app.dev/docs/getting-started/>
- [10] Getting Started · Jest. (s. f.). Jest. <https://jest-bot.github.io/jest/docs/getting-started.html>
- [11] Hudson, P. (2016). Hacking with REACT [Libro electrónico]. <http://www.hackingwithREACT.com/>
- [12] JAVASCRIPT. (2020, 11 agosto). Documentación web de MDN. <https://developer.mozilla.org/es/docs/Web/JAVASCRIPT>
- [13] 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>
- [14] Mobile-first indexing best practices. (s. f.). Google Developers. <https://developers.google.com/search/mobile-sites/mobile-first-indexing>
- [15] Open Source Initiative. (2006). The MIT license. 2015b.[Online]. Available: <https://opensource.org/licenses/MIT>. [Accessed 27 March 2017].
- [16] Popoter Pérez, G. J. (2016). Rediseño de aplicaciones utilizando las tecnologías modernas para el desarrollo web en su parte Front-end.
- [17] 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/>

- [18] ¿Qué es el open source? (2020). Red Hat. <https://www.redhat.com/es/topics/open-source/what-is-open-source>
- [19] REACT – A JAVASCRIPT library for building user interfaces. (s. f.). REACT. <https://REACTjs.org>
- [20] Riquelme, R. (2016, 17 octubre). Los 10 mejores trabajos en TI e Ingeniería. El Economista. <https://www.economista.com.mx/tecnologia/Los-10-mejores-trabajos-en-TI-e-Ingenieria-20161017-0053.html>
- [21] Rodriguez, M. A. (2019, 3 diciembre). Crear y publicar un paquete NPM en 4 sencillos pasos. Medium. <https://medium.com/200-response/crear-y-publicar-un-paquete-npm-en-4-sencillos-pasos-8e7498b558a2>
- [22] Ruiz Inoue, K. (2020, 20 agosto). ¿Qué es el Desarrollo Front end, Back end y Fullstack? Medium. <https://medium.com/deuk/qu%C3%A9-es-el-desarrollo-front-end-back-end-y-fullstack-2941b51af1>
- [23] Stack Overflow Developer Survey 2020. (2020). Stack Overflow. <https://insights.stackoverflow.com/survey/2020overview>
- [24] 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/>
- [25] What is npm. (2020). w3schools. [https://www.w3schools.com/whatis/whatis\\_npm.asp](https://www.w3schools.com/whatis/whatis_npm.asp)
- [26] What is Babel? · Babel. (2020). Babel. <https://babeljs.io/docs/en/>

# Capítulo 12

## Anexos