**Robert Salazar Arias** 11rsahome@gmail.com

## The main objetives in this Project

- Understand how to develop an app with React
- Improve your web development skills
- Improve your JavaScript development skills

## Table of contents

## Phase I - Project Initiation

### Project requirements

We'll use a repository to make the pill

**Responsibility** >https://code.assemblerschool.com/mike/react-basics-1-pill.git
This pill was started with the Create React App package so that you can run the usual scripts to install and run the packages.
**npm install** install packages once you have cloned the repository
**npm run start** to start the development server

### Project specifications

Create a clear and orderly directory structure

Both the code and comments must be written in English.

Use the camelCase code style to define variables and functions

When using HTML, never use inline styles

If you use different programming languages, always define the implementation in separate terms

Remember that it is important to divide tasks into several subtasks so that you can associate each particular step of the construction with a specific commitment

You should try as much as possible that the confirmations and planned tasks are the same

Delete unused or unused files that are not needed to evaluate the project

You must extract and modulate all UI elements to react components

Products must be rendered dynamically using JavaScript loops

Products must be presented as React components

You can't use external state management libraries, just React Hooks

## Phase II - Project Planning

### Reasoning

### 2.1 **Clone the repository**

- o This pill is created so that you can run the scripts to install and run the packages
- or npm install install packages that have been cloned the repository
- or npm run star start the development server.7

### 2.2 **React component**

- o Convert all App.js code to components so that the code is more reusable and modularized possible.
- o Each interface element must be a React component, for example, buttons are UI elements that are reused in various places in the application, so they must be extracted from the React components.

### 2.3 **App Features**

- o Once you've modularized your application into components, you'll need to implement the logic so that you can create an e-commerce application.

### 2.4 **Rendering the products**

- o Products are stored in the products.js file that you must use to dynamically render products on the screen using a JavaScript loop.
- o Each product must have event listeners and the methods needed to handle the necessary UI interactions, such as adding the product to the cart.

### 2.5 **Add to cart**

- o When you click the button you will need to add to the shopping cart
- o By default it will be empty

### 2.5 **Shopping Cart**

Once a product has been added to the cart:

- o Edit the quantity of the product using the selected item that would use the total price of the cart
- o Remove items from cart that will update the total price
- o The total price of the cart must always be updated to present the total cost of all items in the cart.
- o When the cart is empty it must present a message inside the cart that is empty.

### 2.6 **State Administration**

You must use React hooks to handle state management in your application. You cannot use a state management library.

- You should check if the cart item has already been added to your cart to update only the quantity instead of adding it again. The quantity must not exceed 10 units for each product.
- Store items in local storage and load them if the page is reloaded so that cart items are not lost from page refresh
- Each time the page is refreshed, you must upload the items in the local storage cart to save them in the React state, so that the app is displayed with the contents of the local storage, if any. Otherwise, the cart should display the default message of "Your cart is empty"

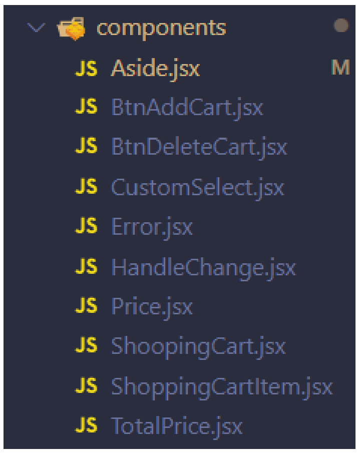
## 3 Implementation

**Clone the repository**

npm install

**Responsibility** >https://code.assemblerschool.com/mike/react-basics-1-pill.git

**React component**
It has been divided into components



```
components
  Aside.jsx            M
  BtnAddCart.jsx
  BtnDeleteCart.jsx
  CustomSelect.jsx
  Error.jsx
  HandleChange.jsx
  Price.jsx
  ShoopingCart.jsx
  ShoppingCartItem.jsx
  TotalPrice.jsx
```

- **Características de la aplicación**

Se ha implementado en para diferentes componentes

**Renderizando los productos**

Los productos se ha renderizado correctamente con use state dinámicamente

```
   import prod from "./products";
```

```
   const [products, saveProduct] = useState(prod);
```

```
//guardar en local Storage
  let prodLocalStorage = JSON.parse(localStorage.getItem("cart"));
  if (!prodLocalStorage) {
    prodLocalStorage = [];
  }
  //Json lo pasamos useState lo metemos a la variable products
  const [cart, addCart] = useState(prodLocalStorage);
  //USamos useEffect saber que se actualiza y mantenerlo
  useEffect(() => {
    let prodLocalStorage = JSON.parse(localStorage.getItem("cart", "count"));
    if (prodLocalStorage) {
      localStorage.setItem("cart", JSON.stringify(cart));
    } else {
      localStorage.setItem("cart", JSON.stringify([]));
    }
  }, [cart]);
```

- **Añadir al carrito**

Al dar clic se ha añae al carrito en el componente de

**BtnAddCart.jsx con esta función pasando los props correspondiente**

```
const handleAddToCart = (id) => {
    const product = products.filter((product) => product.id === id)[0];
    addCart([...cart, product]);
  };
```

Desde Shopping Cart

```
<BtnAddCart
            key={id}
            product={product}
            cart={cart}
            products={products}
            addCart={addCart}
          />
```

HandleChange lo creamos en el aside

```
const handleChange = (e) => {
    e.preventDefault();
    saveCount(parseInt(e.target.value, 10));
    if (count < 1 || count > 9 || isNaN(count)) {
      saveError(true);
      return;
    }
    saveError(false);
  };
```

Nuestro btn delete para eleminar una producto

```
const handleRemove = (id) => {
    const products = cart.filter((product) => product.id !== id);
    addCart(products);
  };
  return (
    <div className="col col-6 col-lg-8">
      <button
        type="btn"
        className="btn btn-dark"
        onClick={() => handleRemove(product.id)}
      >
        Remove
      </button>
    </div>
```

Custom Select para tomar los componentes de btn delete y handlechange pasando un hook que hemos creado en el Aside  que se llama error como variable aplicando una condicion s

```
{error ? (
          <Error message="el valor es incorrecto maximo hasta 10" />
        ) : null}
```

Nuestro componente Error  pasando como mensaje

```
const Error = ({ message }) => {
  return <p className="error text-danger"> {message} </p>;
};
```

Nuestro componente handle Change pasando la funcion que se  ha creado un hook en el aside

```
const HandleChange = ({ handleChange }) => {
  return (
    <div className="col col-6 col-lg-4">
      <input
        type="Number"
        defaultValue="1"
        className="form-control"
        onChange={handleChange}
      />
    </div>
  );
};
```
Nuestro componente price recibiendo props de product y count que se ha creado en el aside mostrado el valor total del producto hay que automatizarlo un poco mas

```
const Price = ({ product, count }) => {
  let result = product.price * count;
  return (
    <Fragment>
      <h4>
        <strong>{result}</strong>
      </h4>
    </Fragment>
  );
};
```
Nuestro componente Shopping cart

Se lo pasamos ala app principal para que reciba los props de los hooks creado en el app principal

```
const ShoopingCart = ({
  product,
  cart,
  addCart,
  products,
  handleChange,
  count,
  error,
}) => {
  const { title, price, img, id } = product;
```
Component Shopping cart item  pasamos los props para utilizarlo en nuestro componentes shooping cart

```
const ShoopingCartItem = ({ cart, addCart, handleChange, count, error }) => {
```

Componente total price los pasamos para utilizar lo enuestro prices

```
const TotalPrice = ({ products, cart, count }) => {
  // console.log(cart)
  return (
    <div className="d-flex justify-content-between">
      <h4 className="h5">Total</h4>
      {cart.map((product) => (
        <Price
          key={product.id}
          products={products}
          cart={cart}
          product={product}
```

```
            count={count}
          />
        ))}
    </div>
```

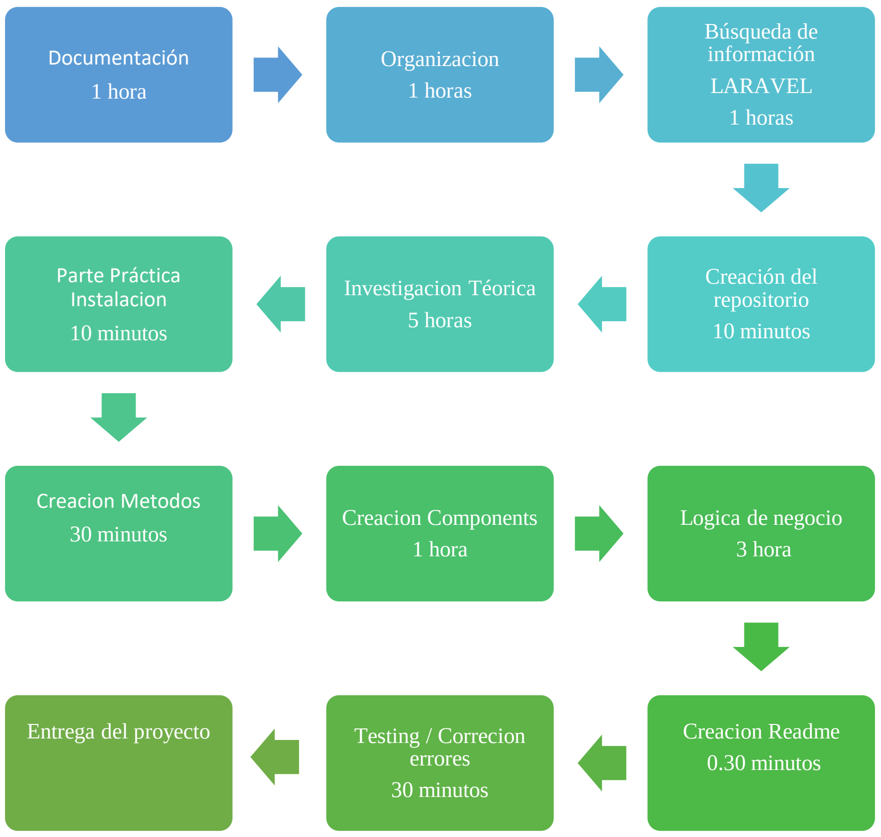
## Planification de Tarea

LISTA DE TAREAS A REALIZAR

| Tarea | Prioridad | Horas | Dificultad | ID |
|---|---|---|---|---|
| Documentación | Alta | 2,00 | Alta | 1 |
| Organización | Alta | 1,00 | Normal | 2 |
| Búsqueda Previa de información | Normal | 1,00 | Normal | 3 |
| Creación de repositorio | Baja | 0,15 | Baja | 4 |
| Investigación Teórica | Alta | 2.00 | Normal | 5 |
| Clonación Repositorio | Baja | 0.15 | Normal | 6 |
| Implementación Componentes | Alta | 2-00 | Alta | 7 |
| Creación README | Baja | 0,30 | Baja | 9 |
| Testing / Corrección Errores | Alta | 0,30 | Normal | 10 |
| Entrega de proyecto | Alta | 0.20 | Alta | 11 |

**Comentado [RdSA1]:**

## Calendario seguimiento del proyecto



## Documentación WORKFLOW DE GIT

- Creación Git Hub **https://github.com/robertfox11/TOOL-ReactBasics.git**
- Hacemos commits de la estructura de la página principal.
- Probabilidad de que ocurra 80%
- Impacto en el proyecto 60%
- Posible alternativa (mitigación) Pedir ayuda a compañeros
- Probabilidad de que ocurra 30%
- Impacto en el proyecto 60%
- Posible alternativa (mitigación) Pedir ayuda a compañeros
- No encontrar con facilidad la información relacionada con el proyecto
- Probabilidad de que ocurra 30%
- Impacto en el proyecto 60%
- Posible alternativa (mitigación)
- Pedir ayuda a compañeros

A partir de la realización de la estructura se continuó trabajando solamente en la rama "master", a través del Workflow "Gitflow".
Mas información --> https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow

## Herramientas

Se utilizaron diferentes herramientas en el desarrollo del proyecto. Son los siguientes:

- ***git: un potente sistema de control de*** versiones que ayuda a realizar un seguimiento de los cambios en el árbol de trabajo.

- ***Visual Studio Code: un editor de*** código optimizado para crear y depurar aplicaciones web modernas.
- ***React,****Components*
- ***Herramientas para desarrolladores de Google Chrome:*** se utiliza para depurar el código JavaScript y para probar los ajustes de diseño.
- ***Documentos de Google:*** se utiliza para escribir la documentación del proyecto.
- ***Validador W3C:*** utilizado para validar el código HTML y CSS.
- ***ESLint:*** utilizado para validar el código JavaScript.

## Fase III - Ejecución del proyecto

### Conceptos

### Incidentes

¡Ninguno, por suerte!

### Lessons

Todas las tareas se completaron sin tener que hacer frente a ningún obstáculo importante.

## Fase IV - Cierre del proyecto

### Comentarios generales

La píldora se completó con éxito en el intervalo de tiempo que se predijo en planificación de tareas.