QuickCyber Store

Documentación de Software

Marcos Damián Pool Canul
Alexis Rodriguez Ramon
Profesor: Ricardo Armando Ruíz Hernández

Presentado para la materia de Diseño de interfaces humano-maquina en cumplimiento al primer proyecto parcial.

Ing. Datos e Inteligencia Organizacional
Universidad del Caribe

19 de FEBRERO DE 2024
[Revised May 13, 2024]

Table of contents

Chapter 1

Introducción

1.1 Contexto del Proyecto

El proyecto "QuickCyber Store" es una aplicación web de comercio electrónico diseñada para ofrecer a los usuarios una experiencia de compra en línea conveniente y eficiente. La idea detrás de esta plataforma es proporcionar a los clientes una amplia variedad de productos de electronica con precios competitivos y una interfaz de usuario intuitiva.

El comercio electrónico ha experimentado un crecimiento significativo en los últimos años, y la comodidad de comprar en línea se ha convertido en una parte integral de la vida cotidiana de muchas personas. Con la creciente demanda de soluciones de comercio electrónico, este proyecto busca ofrecer una experiencia de compra en línea excepcional que sea accesible para todos.

1.2 Objetivos del Proyecto

El objetivo principal de "QuickCyber Store" es proporcionar a los usuarios una plataforma en línea fácil de usar para comprar una amplia gama de productos. Algunos de los objetivos específicos del proyecto incluyen:

- Desarrollar una interfaz de usuario atractiva y fácil de navegar.
- Ofrecer una selección diversa de productos de alta calidad.
- Implementar un carrito de compras y un sistema de pago seguro.
- Proporcionar información detallada sobre cada producto, incluyendo imágenes y descripciones.
- Permitir a los usuarios registrarse, iniciar sesión y realizar un seguimiento de sus pedidos.

1.3 Estructura del Documento

Esta documentación proporciona una visión general detallada de "QuickCyber Store". A lo largo de este informe, se describen los aspectos técnicos, funcionales y de diseño de la aplicación.

El documento se divide en varias secciones, cada una de las cuales aborda un aspecto específico del proyecto. A continuación, se presenta un resumen de las secciones principales:

- **Requisitos del Proyecto:** Esta sección describe los requisitos funcionales y no funcionales del proyecto, incluyendo los casos de uso y los criterios de aceptación.
- **Arquitectura del Sistema:** Aquí se presenta la arquitectura de la aplicación, incluyendo la elección de tecnologías y herramientas utilizadas en el desarrollo.
- **Diseño de la Interfaz de Usuario:** Se describe el diseño de la interfaz de usuario, incluyendo wireframes, esquemas de color y decisiones de diseño.
- **Implementación:** Esta sección aborda la implementación técnica de la aplicación, incluyendo la estructura del código y las características clave.
- **Pruebas y Validación:** Describe las pruebas realizadas para garantizar la calidad y el rendimiento de la aplicación.
- **Conclusiones y Futuras Mejoras:** Se presentan las conclusiones finales del proyecto

y se discuten las posibles mejoras y expansiones futuras.

- **Contacto:** Proporciona información de contacto para los responsables del proyecto.

Esperamos que esta documentación sea una referencia valiosa para comprender y utilizar "QuickCyber Store". Si tienes alguna pregunta o comentario, no dudes en ponerte en contacto con nosotros a través de las direcciones de correo electrónico proporcionadas en la sección de contacto.

Chapter 2

Requisitos del Sistema

Para poder ejecutar correctamente la aplicación *QuickCyber Store*, asegúrate de tener instaladas las siguientes dependencias:

- `"@emotion/react": "^11.11.3"`
- `"@emotion/styled": "^11.11.0"`
- `"@mui/icons-material": "^5.15.7"`
- `"@mui/joy": "^5.0.0-beta.25"`
- `"@mui/material": "^5.15.7"`
- `"@mui/styled-engine-sc": "^6.0.0-alpha.14"`
- `"@testing-library/jest-dom": "^5.17.0"`
- `"@testing-library/react": "^13.4.0"`
- `"@testing-library/user-event": "^13.5.0"`
- `"accounting": "^0.4.1"`
- `"firebase": "10.8.0",`
- `"react": "^18.2.0"`
- `"react-dom": "^18.2.0"`
- `"react-router-dom": "^6.22.0"`
- `"react-scripts": "5.0.1"`
- `"react-uuid": "^2.0.0"`
- `"styled-components": "^6.1.8"`
- `"web-vitals": "^2.1.4"`

Chapter 3

Instalación

Para clonar el repositorio y ejecutar la aplicación en tu máquina local, sigue los siguientes pasos:

1. Clona el repositorio desde GitHub en tu máquina local utilizando el siguiente enlace: `https://github.com/marcosd59/quick-cyber-store`.
2. Abre una terminal en el directorio del proyecto clonado.
3. Ejecuta el comando `npm install` para instalar todas las dependencias requeridas.
4. Una vez completada la instalación, ejecuta el comando `npm start` para iniciar la aplicación en modo de desarrollo.

Chapter 4

Arquitectura

La aplicación *QuickCyber Store* sigue una arquitectura de cliente-servidor típica para aplicaciones web. A continuación, se presenta una descripción general de la arquitectura utilizada:

4.1 Cliente

El cliente de *QuickCyber Store* está construido con React, una biblioteca de JavaScript de código abierto desarrollada por Facebook. React se utiliza para crear una interfaz de usuario interactiva y dinámica que permite a los usuarios interactuar con la aplicación de forma eficiente.

4.2 Servidor

El servidor de *QuickCyber Store* se encarga de manejar las solicitudes del cliente y gestionar los datos de la aplicación. Está construido utilizando Node.js, un entorno de ejecución de JavaScript del lado del servidor que permite ejecutar código JavaScript fuera del navegador web.

4.3 Base de Datos

La base de datos de *QuickCyber Store* almacena toda la información relacionada con los productos, usuarios, pedidos y otra información relevante para el funcionamiento de la aplicación. Se utiliza Firebase para proporcionar flexibilidad y escalabilidad en el almacenamiento de datos.

Chapter 5

Uso

En esta sección, se proporciona una guía sobre cómo utilizar la aplicación *QuickCyber Store*.

5.1 Explorar Productos

Para explorar los productos disponibles en *QuickCyber Store*, sigue estos pasos:

1. Abre el navegador web de tu elección.
2. Dirígete a la URL de la aplicación: `https://quick-cyber-store.com` o `http://localhost:3000/`.
3. Navega por las diferentes categorías de productos utilizando la barra de navegación o la barra de búsqueda.
4. Haz clic en un producto para ver más detalles, incluyendo su descripción, precio y disponibilidad.

5.2 Agregar Productos al Carrito

Una vez que hayas encontrado un producto que te interese, puedes agregarlo a tu carrito de compras:

1. Desde la página principal del producto, haz clic en el botón "Agregar al carrito".
2. El producto se añadirá automáticamente a tu carrito de compras y podrás verlo en la página del carrito.

5.3 Realizar Compra

Para realizar una compra en *QuickCyber Store*, sigue estos pasos:

1. Navega hasta la página del carrito haciendo clic en el icono del carrito en la esquina superior derecha de la pantalla.
2. Verifica los productos en tu carrito y ajusta las cantidades si es necesario.
3. Haz clic en el botón "Pagar" para proceder al proceso de pago.

4. Proporciona la información de envío y de pago requerida.
5. Revisa tu pedido y confirma la compra (No implementado).

5.4 Gestión de Cuenta

QuickCyber Store también ofrece funcionalidades de gestión de cuenta para los usuarios registrados:

1. Inicia sesión en tu cuenta utilizando tus credenciales.
2. Desde tu cuenta, podrás ver tu historial de pedidos, actualizar tu información personal y gestionar tus preferencias de cuenta.

¡Disfruta de tu experiencia de compra en *QuickCyber Store*!

Chapter 6

Desarrollo

6.1 Inicializar plantilla React

Al iniciar el proyecto de QuickCyber Store, se utilizó la plantilla de React proporcionada por Create React App para establecer rápidamente la estructura inicial del proyecto. Create React App es una herramienta popular que simplifica el proceso de configuración y desarrollo de aplicaciones React al proporcionar un entorno de desarrollo preconfigurado y optimizado.

Una vez que se ejecuta el comando para inicializar la plantilla de React, se genera una estructura de directorios básica que incluye archivos como `index.html`, `App.js`, `index.js`, entre otros. Estos archivos proporcionan la base sobre la cual se construirá la aplicación, y se pueden personalizar y expandir según sea necesario para satisfacer los requisitos del proyecto.

1. Creación del proyecto React: Se inicializó un nuevo proyecto React utilizando el comando `npx create-react-app quick-cyber-store`. Esto generó una estructura inicial de archivos y carpetas para el proyecto.
2. Limpieza de archivos innecesarios: Se eliminaron los archivos y carpetas innecesarios que vienen por defecto en un proyecto React recién creado. Esto incluyó la eliminación de logotipos, archivos de prueba y estilos CSS predefinidos.
3. Instalación de dependencias necesarias: Se instalaron las dependencias necesarias para el proyecto, como Material-UI y otras bibliotecas específicas requeridas para la aplicación.
4. Implementación de la plantilla de Material-UI: Se utilizó una plantilla prediseñada de Material-UI para crear componentes visuales, como la barra de navegación (*Navbar*) y las tarjetas de productos (*Product Cards*). Estas plantillas se adaptaron y personalizaron según las necesidades del proyecto.
5. Configuración de la estructura del proyecto: Se definió la estructura general del proyecto, incluyendo la organización de componentes, páginas y datos estáticos. También se planificó el flujo de la aplicación y la navegación entre diferentes vistas.

6.2 Comenzar el proyecto

Una vez que se ha configurado la estructura inicial del proyecto utilizando la plantilla de React y se han realizado las configuraciones necesarias, el siguiente paso es iniciar el servidor de desarrollo. Esto se logra ejecutando el comando `npm start` o `yarn start` en la terminal.

Al ejecutar este comando, se inicia un servidor local que sirve la aplicación React y se abre automáticamente en el navegador predeterminado. Este servidor de desarrollo proporciona actualizaciones en tiempo real, lo que significa que cualquier cambio realizado en el código se reflejará automáticamente en el navegador sin necesidad de recargar la página.

Además de permitir la visualización de la aplicación en el navegador, iniciar el servidor de desarrollo también proporciona un entorno de desarrollo cómodo y optimizado. Esto incluye herramientas como la recopilación y optimización automática del código, la detección de errores y la integración con extensiones de navegador para facilitar la depuración.

6.3 Esquema de la estructura del ecommerce

Se explica la estructura básica del comercio electrónico, incluyendo las páginas principales como la página de inicio, la página del carrito de compras y la página de pago.

El comercio electrónico, o eCommerce, se compone de varias páginas y componentes que interactúan entre sí para ofrecer una experiencia de compra completa.

1. **Página de inicio (Homepage)**: La página de inicio es la puerta de entrada principal del eCommerce. Aquí, los usuarios pueden encontrar una visión general de los productos y servicios ofrecidos por la tienda. Por lo general, incluye elementos como productos destacados, promociones, categorías de productos y enlaces a otras secciones del sitio.
2. **Página del carrito de compras (Shopping Cart Page)**: La página del carrito de compras es donde los usuarios pueden ver los productos que han agregado a su carrito y realizar cambios, como ajustar la cantidad de productos o eliminar elementos. También muestra el resumen del pedido, incluyendo el total de la compra y las opciones de pago.
3. **Página de pago (Checkout Page)**: La página de pago es donde los usuarios completan el proceso de compra. Aquí, ingresan su información de envío y pago, seleccionan el método de envío y revisan el resumen final del pedido antes de confirmarlo. Esta página también puede incluir opciones para aplicar cupones de descuento o utilizar tarjetas regalo.

Estas son solo algunas de las páginas principales que se encuentran en un sitio de comercio electrónico.

6.4 Material-UI

Se elige un diseño de una card de Material-UI como base para los productos y se copia y pega el código para modificarlo posteriormente.

A continuación se muestra el código de la tarjeta de Material-UI que se utilizó como base para los productos:

```
// Código de la tarjeta de Material-UI
import React from 'react';
import { makeStyles } from '@material-ui/core/styles';
import Card from '@material-ui/core/Card';
import CardActionArea from '@material-ui/core/CardActionArea';
import CardActions from '@material-ui/core/CardActions';
import CardContent from '@material-ui/core/CardContent';
import CardMedia from '@material-ui/core/CardMedia';
import Button from '@material-ui/core/Button';
import Typography from '@material-ui/core/Typography';

const useStyles = makeStyles({
  root: {
    maxWidth: 345,
  },
  media: {
    height: 140,
  },
});

export default function ProductCard() {
  const classes = useStyles();

  return (
    <Card className={classes.root}>
      <CardActionArea>
        <CardMedia
          className={classes.media}
          image="/path/to/image.jpg"
          title="Product Image"
        />
        <CardContent>
          <Typography gutterBottom variant="h5" component="h2">
            Product Title
          </Typography>
          <Typography variant="body2" color="textSecondary" component="p">
            Product Description
          </Typography>
```

```
        </CardContent>
      </CardActionArea>
      <CardActions>
        <Button size="small" color="primary">
          Add to Cart
        </Button>
        <Button size="small" color="primary">
          Learn More
        </Button>
      </CardActions>
    </Card>
  );
}
```

Este código proporciona la estructura básica de una tarjeta de producto utilizando los componentes de Material-UI. Posteriormente, este código puede ser modificado y personalizado según las necesidades del proyecto, como cambiar la imagen, el título y la descripción del producto, así como agregar funcionalidades adicionales a los botones.

6.5 Instalar paquetes de Material-UI

Se instalan los paquetes necesarios de Material-UI para utilizar los componentes y estilos en la aplicación.

Para utilizar los componentes y estilos de Material-UI en la aplicación, es necesario instalar los paquetes necesarios. Material-UI es una biblioteca de componentes de React que implementa el diseño y los principios de Material Design de Google.

Los paquetes de Material-UI se pueden instalar fácilmente utilizando un administrador de paquetes como npm o yarn.

```
npm install @material-ui/core
```

Este comando instala el paquete principal de Material-UI que incluye todos los componentes básicos y estilos. Además, si se requieren iconos, se puede instalar el paquete de iconos de Material-UI:

```
npm install @material-ui/icons
```

Una vez instalados estos paquetes, los componentes y estilos de Material-UI estarán disponibles para su uso en la aplicación. Es importante verificar la documentación de Material-UI para obtener más información sobre cómo utilizar los diferentes componentes y personalizar los estilos según sea necesario.

6.6 Personalización del código de la card

Se comienza a personalizar el código de la card para adaptarlo a los productos de QuickCyber Store.

Para adaptar la apariencia de la tarjeta (card) de Material-UI a los productos de QuickCyber Store, se procede a personalizar el código de la tarjeta según las necesidades del proyecto. La tarjeta es un componente visual importante que muestra la información de cada producto de manera atractiva y organizada.

El código de la tarjeta se encuentra en el archivo `Product.js` dentro del directorio de componentes. Este archivo contiene el código base de la tarjeta, que se ha copiado de la documentación de Material-UI y se modificará para reflejar las características específicas de los productos de QuickCyber Store.

La personalización del código de la tarjeta puede incluir cambios en la estructura HTML, estilos CSS, y la integración de datos dinámicos de los productos, como el nombre, precio, imagen, descripción, etc.
Algunos de los elementos que se pueden personalizar en la tarjeta incluyen:

- Ajustes en el diseño y disposición de los elementos.
- Personalización de los estilos CSS para que coincidan con la identidad visual de QuickCyber Store.
- Integración de datos dinámicos de productos para que la tarjeta muestre información relevante para cada producto.

6.7 Librería Accounting npm

Se utiliza la librería Accounting npm para formatear los números en precios con el símbolo de la moneda correspondiente. Esta librería proporciona una manera sencilla y eficiente de formatear números según el estándar de contabilidad, lo que resulta útil para aplicaciones de comercio electrónico donde se necesitan mostrar precios de productos de manera legible y consistente.

El proceso de uso de la librería Accounting npm implica los siguientes pasos:

1. **Instalación de la librería**: Se instala la librería Accounting npm en el proyecto utilizando el administrador de paquetes npm.
2. **Importación del módulo**: Se importa el módulo necesario de la librería Accounting npm en el archivo de JavaScript donde se requiere formatear los números.
3. **Formateo de números**: Se utiliza la función proporcionada por la librería para formatear los números según el formato deseado, incluyendo el símbolo de la moneda correspondiente.
4. **Integración en la aplicación**: Se integra el formateo de números en la aplicación, asegurándose de aplicarlo en los lugares adecuados donde se muestran los precios.

6.8 Implementar un navbar con Material-UI

Se crea un navbar utilizando los componentes de Material-UI y se personaliza para que tenga un diseño único.

A continuación se muestra el código del Navbar de Material-UI que se utilizó como base para los productos:

```
import * as React from 'react';
import AppBar from '@mui/material/AppBar';
import Box from '@mui/material/Box';
import Toolbar from '@mui/material/Toolbar';
import Typography from '@mui/material/Typography';
import Button from '@mui/material/Button';
import IconButton from '@mui/material/IconButton';
import MenuIcon from '@mui/icons-material/Menu';

export default function ButtonAppBar() {
  return (
    <Box sx={{ flexGrow: 1 }}>
      <AppBar position="static">
        <Toolbar>
          <IconButton
            size="large"
            edge="start"
            color="inherit"
            aria-label="menu"
            sx={{ mr: 2 }}
          >
            <MenuIcon />
          </IconButton>
          <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
            News
          </Typography>
          <Button color="inherit">Login</Button>
        </Toolbar>
      </AppBar>
    </Box>
  );
}
```

Para implementar el navbar con Material-UI, seguimos los siguientes pasos:

1. Creamos un nuevo archivo en la carpeta de componentes y lo llamamos ”navbar.js”.
2. Utilizamos el componente AppBar de Material-UI como base para el navbar.
3. Importamos el logo de la empresa y lo añadimos al navbar.
4. Personalizamos el estilo del navbar utilizando el sistema de estilos de Material-UI.

5. Añadimos elementos como el botón de inicio de sesión, el carrito de compras y cualquier otro elemento necesario para la navegación.
6. Utilizamos el componente Button de Material-UI para los botones de inicio de sesión y el carrito de compras.
7. Implementamos funcionalidades adicionales, como el número de elementos en el carrito utilizando el componente Badge de Material-UI.
8. Ajustamos el diseño del navbar para que sea responsive utilizando el sistema de Grid de Material-UI.
9. Probamos el navbar en diferentes tamaños de pantalla para asegurarnos de que se vea bien en todos ellos.

Con estos pasos, logramos implementar un navbar funcional y estilizado utilizando los componentes y estilos proporcionados por Material-UI. Esto nos permite crear un diseño de navbar profesional y adaptativo para nuestra aplicación web.

6.9 Página web responsive con GRID de Material-UI

Se utiliza el sistema de grid de Material-UI para crear una página web responsive con breakpoints para diferentes tamaños de pantalla.

A continuación se muestra el código del Grid de Material-UI que se utilizó como base para los productos:

```
import * as React from 'react';
import { styled } from '@mui/material/styles';
import Box from '@mui/material/Box';
import Paper from '@mui/material/Paper';
import Grid from '@mui/material/Grid';

const Item = styled(Paper)(({ theme }) => ({
  backgroundColor: theme.palette.mode === 'dark' ? '#1A2027' : '#fff',
  ...theme.typography.body2,
  padding: theme.spacing(1),
  textAlign: 'center',
  color: theme.palette.text.secondary,
}));

export default function BasicGrid() {
  return (
    <Box sx={{ flexGrow: 1 }}>
      <Grid container spacing={2}>
        <Grid item xs={8}>
          <Item>xs=8</Item>
        </Grid>
        <Grid item xs={4}>
          <Item>xs=4</Item>
```

```
        </Grid>
        <Grid item xs={4}>
          <Item>xs=4</Item>
        </Grid>
        <Grid item xs={8}>
          <Item>xs=8</Item>
        </Grid>
      </Grid>
    </Box>
  );
}
```

Para crear una página web responsive utilizando el sistema de grid de Material-UI, seguimos estos pasos:

1. Utilizamos el componente Grid de Material-UI para estructurar el diseño de la página.
2. Definimos los breakpoints para diferentes tamaños de pantalla, como xs, sm, md, lg y xl, utilizando las propiedades proporcionadas por el componente Grid.
3. Añadimos los elementos de la página dentro de los componentes Grid, especificando las propiedades como el tamaño de las columnas y filas, la alineación y el espaciado entre los elementos.
4. Personalizamos el diseño de la página utilizando el sistema de grid de Material-UI para asegurarnos de que se vea bien en todos los tamaños de pantalla.
5. Probamos la página en diferentes dispositivos y tamaños de pantalla para garantizar que sea completamente responsive y se adapte correctamente a cada uno.

Con esto, creamos una página web responsive utilizando el sistema de grid de Material-UI, lo que nos permite diseñar y estructurar el contenido de manera flexible.

6.10 Personalización de cada producto

Creamos un archivo llamado products-data.js que contiene un array con los datos de los productos que se mostrarán en la aplicación. Cada objeto en el array representa un producto y contiene información como el nombre, la descripción, el precio y la imagen del producto.

Utilizamos los datos del array de productos para renderizar los componentes de cada producto en la aplicación. Esto implica utilizar un bucle para iterar sobre el array de productos y renderizar un componente personalizado para cada uno de ellos.

Personalizamos los componentes de cada producto según las necesidades de diseño y funcionalidad. Esto puede incluir el uso de estilos, iconos y otras propiedades para garantizar que cada producto se muestre correctamente en la aplicación y cumpla con los requisitos de diseño.

Probamos la aplicación para asegurarnos de que los componentes de cada producto se mues-

tran correctamente y funcionan según lo esperado en diferentes situaciones y dispositivos. Esto garantiza una experiencia de usuario óptima y coherente en QuickCyber Store.

6.11 Checkout page

Se crea la página del carrito de compras (checkout page) utilizando cards modificadas para permitir la eliminación de productos y el cálculo del precio total.

Para ello, comenzamos importando los datos de los productos desde el archivo Product-Data.js y los utilizamos para renderizar los componentes de cada producto en la página del carrito de compras.

Utilizamos el sistema de grid de Material-UI para diseñar la estructura de la página del carrito de compras de forma que sea responsive, con breakpoints para diferentes tamaños de pantalla.

Personalizamos los componentes de cada producto en la página del carrito de compras para que se ajusten a los requisitos de diseño y funcionalidad, incluyendo la eliminación de productos y el cálculo del precio total.

Finalmente, probamos la funcionalidad de la página del carrito de compras para asegurarnos de que los productos se muestran correctamente y se pueden eliminar, y que el precio total se calcula correctamente.

6.12 Funcionalidad del "total"

Se implementa la funcionalidad para calcular dinámicamente el precio total del carrito de compras. Esta funcionalidad asegura que el precio total se actualice automáticamente cada vez que se agregue o elimine un producto del carrito.

Para lograr esto, se utiliza un estado local para almacenar la lista de productos en el carrito y su precio asociado. Cada vez que se agrega o elimina un producto, se actualiza este estado y se recalcula el precio total en consecuencia.

Además, se utilizan métodos de actualización de estado y efectos secundarios para manejar los cambios en la lista de productos y recalcular el precio total de manera eficiente.

La implementación de esta funcionalidad garantiza una experiencia de usuario fluida y precisa, ya que el precio total se mantiene siempre actualizado y refleja con precisión el costo total de los productos en el carrito.

```
const calculateTotalPrice = () => {
  // Función para calcular el precio total de los productos
  const totalPrice = basket?.reduce(
    (total, product) => total + product.price,
    0
  );
  return totalPrice.toFixed(2); // Redondear a 2 decimales
};
```

6.13 Estructura del eCommerce y Context API

En esta sección, se describe la estructura del eCommerce y cómo se utiliza Context API junto con el archivo 'reducer.js' para gestionar el estado global de la aplicación y el carrito de compras.

Se comenzó estableciendo una infraestructura para Context API, lo cual permite definir una variable, como un array, para almacenar los productos seleccionados. Cada vez que se agrega un producto al carrito, se actualiza este array y otros componentes de la aplicación tienen acceso a esta información.

Se crearon dos archivos externos para Context API: 'reducer.js' y 'stateProvider.js'. El primero contiene la lógica para gestionar el estado global, incluyendo el estado inicial y las acciones que se pueden realizar, como agregar o eliminar elementos del carrito. El segundo proporciona las herramientas necesarias para pasar datos entre componentes.

En 'reducer.js', se definió el estado inicial del carrito como un array vacío y se establecieron las acciones que puede realizar el reductor, como agregar o eliminar elementos del carrito.

En 'stateProvider.js', se creó el contexto que permite compartir el estado entre componentes y se configuró el proveedor de estado para que todos los componentes de la aplicación puedan acceder al estado global.

Se implementó la funcionalidad para agregar y eliminar productos del carrito. Al hacer clic en un producto, se añade al carrito y se actualiza la cantidad mostrada en el ícono del carrito en la barra de navegación. También se añadió la funcionalidad para eliminar productos individuales del carrito.

Finalmente, se implementó la funcionalidad para calcular el total de la compra, sumando los precios de todos los productos en el carrito. Esto se logró mediante el uso de la función 'reduce' de JavaScript.

Esta sección detalla cómo se estructura el eCommerce y cómo se utiliza Context API para gestionar el estado global de la aplicación y el carrito de compras, permitiendo una interacción fluida y coherente para los usuarios.

6.14 Instalación de React Router DOM

En esta sección, se describe la instalación y el uso de React Router DOM para crear una serie de rutas que permiten la navegación dentro de la página web.

Se comenzó instalando React Router DOM mediante el comando 'npm i react-router-dom'. Esta biblioteca proporciona componentes de enrutamiento para React, lo que facilita la creación de rutas y la navegación entre páginas en una aplicación web.

Luego, se configuraron las rutas en la aplicación utilizando los componentes proporcionados por React Router DOM, como 'BrowserRouter', 'Route', 'Switch' y 'Link'.

El componente 'BrowserRouter' se utilizó para envolver toda la aplicación y establecer un contexto de enrutamiento. Dentro de este componente, se definieron las rutas utilizando el componente 'Route', donde se especifica la URL y el componente que se renderizará cuando se visite esa ruta.

El componente 'Switch' se utilizó para envolver varias rutas y garantizar que solo se renderice la primera ruta coincidente. Esto evita que se rendericen múltiples rutas al mismo tiempo.

Además, se utilizaron los componentes 'Link' para crear enlaces de navegación entre diferentes páginas de la aplicación. Estos enlaces permiten a los usuarios hacer clic en ellos para navegar de forma rápida y sencilla.

La instalación y el uso de React Router DOM permiten crear una experiencia de navegación fluida y dinámica en la página web, facilitando la navegación entre diferentes secciones y componentes de la aplicación.

6.15 Funcionalidad de los botones de las cards

Se añade funcionalidad a los botones del icono del carrito de cada card para que al hacer clic se añada el producto al carrito, y al icono de la papelera para que al hacer clic se elimine el producto del carrito.

Botón del icono del carrito: Se agregó funcionalidad para que al hacer clic en este botón, se añada el producto correspondiente al carrito de compras. Esto se logró mediante la implementación de un evento `onClick` que llama a una función `addToCart` y pasa el producto como argumento. Dentro de esta función, se utiliza el contexto de la aplicación para agregar el producto al carrito.

Con esta funcionalidad implementada, los usuarios pueden agregar productos al carrito y eliminarlos fácilmente desde las cards de productos, lo que mejora la experiencia de usuario y hace más intuitiva la gestión de compras en la aplicación.

6.16 Actualización de los ítems

En esta sección, se detalla la implementación de la funcionalidad para actualizar el número de productos en el carrito y la suma de los precios de los diferentes productos.

1. **Actualización del número de productos:** Se agregó lógica para mantener actualizado el número de productos en el carrito. Esto se logró mediante la actualización dinámica del contador que muestra la cantidad de productos en el carrito cada vez que se agrega o elimina un producto.
2. **Actualización de la suma de precios:** Se implementó la funcionalidad para calcular la suma total de los precios de los productos en el carrito. Se utilizó una función de reducción (`reduce`) en JavaScript para sumar los precios de todos los productos en el carrito y mostrar el total actualizado en la interfaz de usuario.

```
export const initialState = {
  basket: [],
  user: null,
};

export const actionTypes = {
  ADD_TO_BASKET: "ADD_TO_BASKET",
  REMOVE_FROM_BASKET: "REMOVE_FROM_BASKET",
  SET_USER: "SET_USER",
  EMPTY_BASKET: "EMPTY_BASKET",
};

const reducer = (state, action) => {
  console.log(action);
  switch (action.type) {
    case "ADD_TO_BASKET":
      return {
        ...state,
        basket: [...state.basket, action.item],
      };
    case "REMOVE_FROM_BASKET":
      const index = state.basket.findIndex(
        (basketItem) => basketItem.id === action.id
      );
      let newBasket = [...state.basket];
      if (index >= 0) {
        newBasket.splice(index, 1);
      } else {
        console.warn(
          `Can't remove product (id: ${action.id}) as its not in basket!`
```

```
        );
      }
      return {
        ...state,
        basket: newBasket,
      };
    case "SET_USER":
      return {
        ...state,
        user: action.user,
      };
    case "EMPTY_BASKET":
      return {
        ...state,
        basket: action.basket,
      };
    default:
      return state;
  }
};

export default reducer;
```

Con estas actualizaciones, los usuarios pueden ver fácilmente la cantidad total de productos en su carrito y el precio total de su compra, lo que mejora la experiencia de compra y proporciona una visión clara de su selección de productos.

6.17 Autenticación

Se añade autenticación al proyecto creando un proyecto en Firebase para permitir a los usuarios registrarse e iniciar sesión.

Para comenzar con la autenticación, se siguieron los siguientes pasos:

1. Se creó un proyecto en Firebase desde la consola de Firebase, nombrándolo "QuickCyberStore".
2. Se omitió la configuración de Google Analytics y se procedió a crear el proyecto.
3. Se accedió a la configuración del proyecto en Firebase para obtener las claves de configuración necesarias para conectar Firebase con el proyecto en Visual Studio Code.
4. Se creó un archivo llamado "firebase.js" en la carpeta "src" del proyecto y se inicializó Firebase con las claves de configuración proporcionadas.
5. Se creó el objeto de autenticación utilizando Firebase Authentication y se exportó para poder utilizarlo en toda la aplicación.
6. Se implementó la página de registro ("signup") utilizando un componente de Material UI predefinido para capturar el correo electrónico y la contraseña del usuario.

7. Se implementó la lógica para registrar un usuario utilizando la función ”createUserWithEmailAndPassword” proporcionada por Firebase Authentication.
8. Se implementó la página de inicio de sesión (”signin”) utilizando un componente de Material UI predefinido para capturar el correo electrónico y la contraseña del usuario.
9. Se implementó la lógica para iniciar sesión utilizando la función ”signInWithEmailAndPassword” proporcionada por Firebase Authentication.
10. Se utilizó el hook useHistory de React Router DOM para redirigir al usuario a la página de inicio después de que se haya registrado o iniciado sesión correctamente.
11. Se agregaron enlaces en la página de registro para permitir a los usuarios navegar fácilmente entre la página de registro y la página de inicio de sesión.
12. Se agregaron enlaces en la página de inicio de sesión para permitir a los usuarios navegar fácilmente entre la página de inicio de sesión y la página de registro.

A continuación se muestra el código del SignIn de Material-UI que se utilizó como base:

```
import * as React from 'react';
import Avatar from '@mui/material/Avatar';
import Button from '@mui/material/Button';
import CssBaseline from '@mui/material/CssBaseline';
import TextField from '@mui/material/TextField';
import FormControlLabel from '@mui/material/FormControlLabel';
import Checkbox from '@mui/material/Checkbox';
import Link from '@mui/material/Link';
import Grid from '@mui/material/Grid';
import Box from '@mui/material/Box';
import LockOutlinedIcon from '@mui/icons-material/LockOutlined';
import Typography from '@mui/material/Typography';
import Container from '@mui/material/Container';
import { createTheme, ThemeProvider } from '@mui/material/styles';

function Copyright(props: any) {
  return (
    <Typography variant="body2" color="text.secondary" align="center" {...props}>
      {'Copyright © '}
      <Link color="inherit" href="https://mui.com/">
        Your Website
      </Link>{' '}
      {new Date().getFullYear()}
      {'.'}
    </Typography>
  );
}

// TODO remove, this demo shouldn't need to reset the theme.
```

```
const defaultTheme = createTheme();

export default function SignIn() {
  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    const data = new FormData(event.currentTarget);
    console.log({
      email: data.get('email'),
      password: data.get('password'),
    });
  };

  return (
    <ThemeProvider theme={defaultTheme}>
      <Container component="main" maxWidth="xs">
        <CssBaseline />
        <Box
          sx={{
            marginTop: 8,
            display: 'flex',
            flexDirection: 'column',
            alignItems: 'center',
          }}
        >
          <Avatar sx={{ m: 1, bgcolor: 'secondary.main' }}>
            <LockOutlinedIcon />
          </Avatar>
          <Typography component="h1" variant="h5">
            Sign in
          </Typography>
          <Box component="form" onSubmit={handleSubmit} noValidate sx={{ mt: 1 }}>
            <TextField
              margin="normal"
              required
              fullWidth
              id="email"
              label="Email Address"
              name="email"
              autoComplete="email"
              autoFocus
            />
            <TextField
              margin="normal"
              required
              fullWidth
```

```
              name="password"
              label="Password"
              type="password"
              id="password"
              autoComplete="current-password"
            />
            <FormControlLabel
              control={<Checkbox value="remember" color="primary" />}
              label="Remember me"
            />
            <Button
              type="submit"
              fullWidth
              variant="contained"
              sx={{ mt: 3, mb: 2 }}
            >
              Sign In
            </Button>
            <Grid container>
              <Grid item xs>
                <Link href="#" variant="body2">
                  Forgot password?
                </Link>
              </Grid>
              <Grid item>
                <Link href="#" variant="body2">
                  {"Don't have an account? Sign Up"}
                </Link>
              </Grid>
            </Grid>
          </Box>
        </Box>
        <Copyright sx={{ mt: 8, mb: 4 }} />
      </Container>
    </ThemeProvider>
  );
}
```

6.18 Implementar formulario

Se elige una plantilla en Material-UI para el formulario de pago y se integra en la página web.

Se describe el proceso de implementación del formulario de pago en el proyecto. Se siguió el siguiente procedimiento:

1. Se seleccionó una plantilla de Material-UI para el formulario de pago que se ajustara a los requisitos del proyecto y proporcionara una experiencia de usuario intuitiva.
2. Se integró la plantilla del formulario de pago en la página web del proyecto.
3. Se realizaron ajustes en la plantilla según las necesidades específicas del proyecto, como la adición de campos adicionales o la modificación del diseño.
4. Se implementó la lógica necesaria para validar los datos ingresados por el usuario en el formulario de pago, asegurando que se proporcionen los datos requeridos y que estén en el formato correcto.
5. Se configuraron las acciones a realizar una vez que el usuario envíe el formulario de pago, como procesar el pago, mostrar un mensaje de confirmación o redirigir a una página de agradecimiento.
6. Se realizó una revisión exhaustiva del formulario de pago para garantizar su correcto funcionamiento y una experiencia de usuario sin problemas.

Con la implementación del formulario de pago, los usuarios pueden completar sus transacciones de manera segura y conveniente en la página web del proyecto.

6.19 Pasarela de Pago

1. **Configuración de Stripe**: Primero, es necesario tener una cuenta en Stripe y obtener las claves API públicas y secretas. Estas claves se encuentran en el Dashboard de Stripe en la sección de Desarrolladores.

2. **Instalación de Stripe**: Para utilizar Stripe en el proyecto, se deben instalar las dependencias necesarias. Esto se puede hacer ejecutando el siguiente comando:

   ```
   \verb|npm install @stripe/react-stripe-js @stripe/stripe-js|
   ```

3. **Configuración del Frontend**: En el frontend, se utiliza el componente PaymentForm para manejar la lógica de pago. Primero, se debe inicializar Stripe con la clave pública:

   ```
   // PaymentForm.jsx
   import { loadStripe } from '@stripe/stripe-js';
   import { Elements } from '@stripe/react-stripe-js';

   const stripePromise = loadStripe(process.env.REACT_APP_STRIPE_PUBLIC_KEY);
   ```

   Luego, se utiliza el componente Elements de Stripe para envolver el formulario de pago, lo que permite usar los elementos de Stripe y el hook useStripe:

   ```
       // PaymentForm.jsx
   const PaymentForm = ({ backStep, nextStep }) => {
     return (
   ```

```
    <>
      <Review />
      <Divider />
      <Typography variant="h6" gutterBottom sx={{ marginTop: "1rem" }}>
        Payment method
      </Typography>
      <Elements stripe={stripePromise}>
        <CheckoutForm backStep={backStep} nextStep={nextStep} />
      </Elements>
    </>
  );
};
```

4. **Creación del Método de Pago**: Dentro del componente CheckoutForm, se maneja la lógica para crear un método de pago utilizando stripe.createPaymentMethod y se envía la información al backend:

```
// PaymentForm.jsx
const handleSubmit = async (e) => {
  e.preventDefault();
  const { error, paymentMethod } = await stripe.createPaymentMethod({
    type: "card",
    card: elements.getElement(CardElement),
  });

  if (!error) {
    const { id } = paymentMethod;
    // Lógica para enviar el ID del método de pago y otros detalles al backend
  }
};
```

5. **Configuración del Backend**: En el backend, se configura una ruta para manejar las solicitudes de pago. Se utiliza la clave secreta de Stripe para crear un paymentIntent:

```
    // index.js
app.post("/api/checkout", async (req, res) => {
  const { id, amount, items, names, shippingData } = req.body;

  try {
    const payment = await stripe.paymentIntents.create({
      amount,
      currency: "MXN",
      description: `Purchase of ${names} with ${items} items. Shipping to ${shippin
      payment_method: id,
      confirm: true,
    });
```

```
    return res.status(200).json({ message: "Payment successful" });
  } catch (error) {
    return res.json({ message: error.raw.message });
  }
});
```

6. **Procesamiento del Pago**: Una vez que el backend recibe la solicitud de pago, crea el paymentIntent y responde al frontend con el resultado del pago. El frontend puede entonces mostrar un mensaje de éxito o error basado en la respuesta.

7. **Variables de Entorno**: Es importante almacenar las claves API de Stripe en variables de entorno para proteger la información sensible. En el frontend, se puede acceder a estas variables usando

   `process.env.NOMBRE_VARIABLE.`

6.20 Deploy en Netlify

1. **Prerrequisitos**: Asegúrate de que tu proyecto esté listo en un repositorio Git (GitHub, GitLab o Bitbucket). Netlify se conecta directamente a estos servicios para obtener tu código.
2. **Crear una Cuenta en Netlify**: Si aún no tienes una, regístrate en `netlify.com`.
3. **Vincular Tu Repositorio**: Una vez que inicies sesión, puedes crear un nuevo sitio vinculando tu repositorio Git. Netlify solicitará permisos para acceder a tu repositorio y habilitar la implementación continua.
4. **Configuración de la Construcción**: Netlify intentará detectar automáticamente tu configuración de construcción. Para una aplicación React, por ejemplo, el comando de construcción suele ser `npm run build` o `yarn build`, y el directorio de publicación suele ser `build/`. Asegúrate de que estas configuraciones coincidan con la configuración de tu proyecto.
5. **Desplegar**: Después de configurar tu construcción, Netlify construirá y desplegará tu sitio. Recibirás una URL única de `netlify.app` que puedes personalizar.
6. **Despliegue Continuo**: Una vez configurado, cada push a tu rama seleccionada (típicamente `main` o `master`) activará una nueva construcción y despliegue en Netlify, manteniendo tu sitio actualizado.

6.21 Configurando Variables de Entorno

1. **Acceder a Configuración del Sitio**: En tu panel de control de Netlify, selecciona tu sitio y ve a "Configuración" > "Construir y desplegar" > "Entorno".
2. **Agregar Variables**: Aquí, puedes agregar pares clave-valor para tus variables de entorno. Por ejemplo, podrías agregar una variable `REACT_APP_API_URL` para almacenar la URL de tu API.

3. **Usar Variables en Tu Aplicación**: En tu código, puedes acceder a estas variables usando `process.env.NOMBRE_DE_VARIABLE`. Por ejemplo, en una aplicación React, `process.env.REACT_APP_API_URL` accedería al valor de `REACT_APP_API_URL`.
4. **Desarrollo Local**: Cuando trabajes localmente, puedes definir variables de entorno en un archivo `.env` en la raíz de tu proyecto. Recuerda, nunca comprometas este archivo en tu repositorio Git (agrega `.env` a tu archivo `.gitignore`).
5. **Nota de Seguridad**: Ten en cuenta la información que almacenas en las variables de entorno, especialmente en el código frontend, ya que puede ser expuesta al navegador. Evita almacenar datos sensibles como claves API que requieran secreto.

1. **Prerrequisitos**: Asegúrate de que tu proyecto esté listo en un repositorio Git (GitHub, GitLab o Bitbucket). Netlify se conecta directamente a estos servicios para obtener tu código.
2. **Crear una Cuenta en Netlify**: Si aún no tienes una, regístrate en `netlify.com`.
3. **Vincular Tu Repositorio**: Una vez que inicies sesión, puedes crear un nuevo sitio vinculando tu repositorio Git. Netlify solicitará permisos para acceder a tu repositorio y habilitar la implementación continua.
4. **Configuración de la Construcción**: Netlify intentará detectar automáticamente tu configuración de construcción. Para una aplicación React, por ejemplo, el comando de construcción suele ser `npm run build` o `yarn build`, y el directorio de publicación suele ser `build/`. Asegúrate de que estas configuraciones coincidan con la configuración de tu proyecto.
5. **Desplegar**: Después de configurar tu construcción, Netlify construirá y desplegará tu sitio. Recibirás una URL única de `netlify.app` que puedes personalizar.
6. **Despliegue Continuo**: Una vez configurado, cada push a tu rama seleccionada (típicamente `main` o `master`) activará una nueva construcción y despliegue en Netlify, manteniendo tu sitio actualizado.

6.22 Configurando Variables de Entorno

1. **Acceder a Configuración del Sitio**: En tu panel de control de Netlify, selecciona tu sitio y ve a "Configuración" > "Construir y desplegar" > "Entorno".
2. **Agregar Variables**: Aquí, puedes agregar pares clave-valor para tus variables de entorno. Por ejemplo, podrías agregar una variable `REACT_APP_API_URL` para almacenar la URL de tu API.
3. **Usar Variables en Tu Aplicación**: En tu código, puedes acceder a estas variables usando `process.env.NOMBRE_DE_VARIABLE`. Por ejemplo, en una aplicación React, `process.env.REACT_APP_API_URL` accedería al valor de `REACT_APP_API_URL`.
4. **Desarrollo Local**: Cuando trabajes localmente, puedes definir variables de entorno en un archivo `.env` en la raíz de tu proyecto. Recuerda, nunca comprometas este archivo en tu repositorio Git (agrega `.env` a tu archivo `.gitignore`).
5. **Nota de Seguridad**: Ten en cuenta la información que almacenas en las variables de entorno, especialmente en el código frontend, ya que puede ser expuesta al navegador.

Evita almacenar datos sensibles como claves API que requieran secreto.

Chapter 7

Conclusión

En esta documentación se ha detallado el proceso de desarrollo de una aplicación web utilizando tecnologías modernas como React, Redux, Firebase y Netlify. A lo largo del proyecto, se han abordado diversos aspectos relacionados con el diseño, implementación y despliegue de la aplicación, lo que ha permitido adquirir un amplio conocimiento sobre el desarrollo web.

Una de las principales lecciones aprendidas durante este proyecto ha sido la importancia de un enfoque modular y escalable en el desarrollo de aplicaciones web. El uso de componentes reutilizables en React y la gestión centralizada del estado con Redux han demostrado ser prácticas efectivas para mejorar la mantenibilidad y la escalabilidad del código.

La integración de Firebase ha añadido una capa adicional de funcionalidades a la aplicación, permitiendo la implementación de características como la autenticación de usuarios y el almacenamiento de datos en tiempo real de manera sencilla y eficiente.

El despliegue en Netlify ha facilitado la entrega continua del proyecto, garantizando una disponibilidad constante del sitio web y simplificando el proceso de actualización y mantenimiento.

Este proyecto ha sido una experiencia enriquecedora que ha permitido aplicar conocimientos teóricos en un entorno práctico y colaborativo. Se espera que esta documentación sirva como una guía útil para futuros proyectos de desarrollo web, proporcionando información detallada sobre las herramientas, técnicas y mejores prácticas utilizadas durante el proceso.

Chapter 8

Contacto

Puedes ponerte en contacto con nosotros a través de las siguientes direcciones de correo electrónico:

- Marcos Damian Pool Canul - 200300591@ucaribe.edu.mx
- Alexis Rodriguez Ramon - 170300123@ucaribe.edu.mx