Tlajomulco es tuyo

MANUAL TÉCNICO

[Cliente]

PROYECTO

PORTAL WEB CÉDULA DE MERCADOS

Elaborado por:

Alvarado Mombela Cristopher Yahir

Fernández Márquez Braulio Israel

Silva Palmas Angélica Araceli

Índice

1. Introducción

1.1 Objetivos del Manual

Este manual tiene como objetivo proporcionar a los desarrolladores que trabajarán en el lado del cliente del proyecto una guía detallada sobre la arquitectura, la estructura del código y las funciones principales.

1.2 Descripción General del Cliente

El cliente está desarrollado utilizando React y Next.js, lo que permite la creación de interfaces de usuario interactivas y una gestión eficiente del enrutamiento.

2. Estructura del Cliente

La parte del cliente está organizada dentro de la carpeta 'client'. A continuación, se detalla la estructura:

client:

- **node_modules:** Carpeta generada automáticamente por npm o yarn que contiene las dependencias del proyecto React. No se debe modificar manualmente.
- **public:** Contiene archivos estáticos que se servirán directamente al cliente, como favicon.png, index.html, y manifest.json.
- **src:** Carpeta principal del código fuente del cliente React.
    - **assest:** Almacena respectivamente archivos de fuentes y archivos de imágenes utilizados en el proyecto.
    - **components:** Contiene componentes reutilizables utilizados en toda la aplicación.
    - **customHooks:** Contiene ganchos personalizados de React diseñados para encapsular lógica reutilizable en la aplicación.
    - **middleware:** Contiene funciones middleware esenciales para gestionar aspectos como la autorización, la seguridad y otros procesos intermedios en la aplicación.
    - **views:** Almacena los componentes React relacionados con las diferentes vistas o páginas de la aplicación.

    - **App.js:** Es el componente principal que actúa como punto de entrada de la aplicación.
    - **index.js:** Archivo de entrada que inicia la aplicación React y conecta con el punto de entrada del DOM.
    - **index.css:** Archivo de estilo principal, este archivo centraliza las definiciones de fuentes personalizadas y configura el uso de Tailwind CSS en el proyecto
- **package.json y package-lock.json:** Archivos que describen el proyecto, sus dependencias y scripts de construcción.
- **tailwind.config.js:** Archivo de la configuración de Tailwind CSS.

3. Herramientas y Tecnologías Utilizadas

Las tecnologías y herramientas clave utilizadas en "Portal Web Cédula de Mercados" para la parte del cliente incluyen:

- **HTML:** Fundamentales para la estructura y el diseño visual de la parte del cliente.
- **JavaScript:** Esencial para agregar interactividad y dinamismo a la interfaz del usuario.
- **React:** Un framework de JavaScript que facilita la creación de componentes reutilizables y el manejo eficiente del estado de la aplicación.
- **Tailwind:** Un framework CSS útil para diseñar y estilizar rápidamente la interfaz del usuario.

4. Detalles de Implementación

En el desarrollo del cliente, se presta especial atención a los detalles de implementación para garantizar un funcionamiento eficiente y una experiencia de usuario óptima. A continuación, se describen algunos aspectos clave de la implementación:

4.1 Archivos Principales

4.1.1 App.js:

El archivo App.js es el componente principal que actúa como punto de entrada de la aplicación. Este componente utiliza el enrutador (BrowserRouter y Routes) de react-router-dom para gestionar la navegación entre diferentes vistas. A continuación, se proporciona una descripción detallada del contenido del archivo:

Importación de Librerías y Componentes:

- Se importan las librerías y componentes necesarios para la configuración de las rutas y las diferentes vistas de la aplicación. Esto incluye componentes para el inicio de sesión, paneles de control, formularios, vistas de PDF, entre otros.

Definición del Componente Funcional App:

- Se define el componente funcional App, que representa la aplicación en su totalidad.
- Se utiliza el hook useState para gestionar el estado de userData, que probablemente almacene información sobre el usuario autenticado.

Configuración de Rutas con BrowserRouter y Routes:

- Se utiliza el componente BrowserRouter para envolver las rutas de la aplicación y habilitar la navegación mediante URL.
- Se define el componente Routes, que contendrá las diferentes rutas de la aplicación.

Definición de Rutas con el Componente Route:

- Se definen las rutas mediante el componente Route. Cada ruta especifica una URL (path) y el componente que debe renderizarse (element) cuando se accede a esa URL.

Uso de Middlewares y Protección de Rutas:

- Se utilizan los componentes LogOut, ExecutiveProtectedRoute, y AdminProtectedRoute para gestionar aspectos como la autenticación y la

protección de rutas. Por ejemplo, se utiliza LogOut para cerrar la sesión del usuario.

Esta estructura permite una gestión organizada de las rutas y garantiza que las vistas asociadas se rendericen de acuerdo con la autenticación del usuario. La aplicación utiliza el enrutador para proporcionar una experiencia de usuario fluida al navegar entre diferentes secciones.

4.1.2 Index.js:

El archivo index.js es el punto de entrada principal de la aplicación React. A continuación, se proporciona una descripción de su contenido:

Importaciones:

- React: Se importa la biblioteca principal de React.
- ReactDOM: Se importa la funcionalidad de ReactDOM, específicamente ReactDOM.createRoot para crear un "root" para renderizar la aplicación.
- Se importa el archivo de estilos (index.css).
- Se importa el componente principal de la aplicación (App).

Renderización de la aplicación:

- Se crea un "root" utilizando ReactDOM.createRoot, que es una característica de ReactDOM para renderizar de manera optimizada.
- Se utiliza el método render del "root" creado para renderizar la aplicación.
- La aplicación (<App />) se envuelve en <React.StrictMode>, que es un componente de React utilizado para detectar posibles problemas en la aplicación durante el desarrollo y en modo de producción.
- Renderización en el DOM: La renderización de la aplicación se realiza en el elemento del DOM con el id root. Se obtiene este elemento utilizando document.getElementById('root').
- React Strict Mode: Se utiliza <React.StrictMode> para activar el modo estricto de React, que ayuda a detectar problemas comunes y posibles errores durante el desarrollo.

Este archivo es esencial para iniciar la aplicación React y establecer la conexión entre el código de React y el DOM. La inclusión de StrictMode ayuda a mejorar la calidad del código y a anticipar posibles problemas. La renderización se realiza en un "root" creado específicamente para lograr una renderización más eficiente y optimizada.

4.1.3 Index.css:

El archivo index.css juega un papel crucial en la apariencia y estilo de la aplicación. A continuación, se detallan las secciones clave de este archivo:

Estilos con Tailwind CSS

- @tailwind base;
- @tailwind components;
- @tailwind utilities;

Estas líneas importan automáticamente la base, los componentes y las utilidades de Tailwind CSS. Tailwind es un marco de trabajo que simplifica el proceso de estilización al proporcionar clases predefinidas que se pueden aplicar directamente a los elementos HTML.

Definición de Fuentes Personalizadas

Aquí se definen fuentes personalizadas utilizando la regla @font-face. Esto permite el uso de fuentes específicas en la aplicación. Por ejemplo:

```
@font-face {
    font-family: 'Foco-Corp-Bold';
    src: url('../src/assets/fonts/Foco-Corp-Bold.ttf');
}
```

4.1.4 Tailwind.config.js:

El archivo tailwind.config.js es parte de la configuración de Tailwind CSS, una herramienta popular para el desarrollo de estilos en aplicaciones web. A continuación, se proporciona una descripción del contenido del archivo:

Configuración de Contenido:

- **content**: Se especifica la ubicación de los archivos que Tailwind debe analizar para generar estilos. En este caso, se están incluyendo todos los archivos con extensiones .js, .jsx, .ts, y .tsx dentro de la carpeta src.

Configuración del Tema:

- **theme**: Aquí se extiende o personaliza el tema predeterminado de Tailwind. Algunas personalizaciones notables incluyen:
  - Definición de colores personalizados bajo la propiedad colors.
  - Extensión de fuentes personalizadas bajo la propiedad fontFamily.
  - Definición de estilos comunes para los inputs bajo la propiedad "input-common".
- **Plugins**:
  - plugins: Se deja vacío en este caso, pero es el lugar donde se pueden agregar plugins adicionales para Tailwind.

Este archivo permite personalizar la configuración de Tailwind CSS según las necesidades del proyecto. En este caso, se están agregando colores personalizados, fuentes personalizadas y estilos comunes para los inputs. Además, se especifica qué archivos se deben analizar para generar los estilos. Tailwind utiliza esta configuración para generar un archivo de estilos optimizado basado en las clases que realmente se utilizan en el código fuente de la aplicación.

4.2 Carpetas Adicionales

4.2.1 Carpeta customHooks

La carpeta customHooks almacena ganchos personalizados de React que encapsulan lógica reutilizable en la aplicación. Estos ganchos están diseñados para mejorar la modularidad y la reutilización del código al separar la lógica del componente.

Custom Hook- UseParamsDataComerciante.js:

Este hook personalizado, useParamsDataComerciante, proporciona un conjunto de estados y funciones para gestionar la lógica relacionada con los parámetros y datos de los comerciantes en la aplicación. Aquí hay una descripción detallada de los elementos clave de este hook:

Tabs y Estados Asociados:

- activeTab: Estado que almacena la pestaña activa actualmente.
- Estados específicos para la pestaña 1 (Tab1) y la pestaña 2 (Tab2):
  - isMenuOpen: Indica si el menú está abierto.
  - showButtons: Controla la visibilidad de los botones.
  - fieldsEditable: Indica si los campos son editables.

Datos del Comerciante:

- data: Almacena los datos modificados por el usuario.
- originalData: Almacena los datos originales sin modificaciones.
- telefonos: Almacena el estado inicial de los teléfonos.
- telefono1 y telefono2: Almacenan los números de teléfono que se muestran en los campos respectivos.
- fecha_inicio y fecha_termino: Almacenan las fechas de inicio y término del permiso.

Selecciones:

- selectedClasificacion, selectedHorario, y selectedTipo: Almacenan las opciones seleccionadas para clasificación, horario y tipo, respectivamente.

Funciones de Modificación de Estado:

- Funciones asociadas a cada estado para actualizar sus valores.

Uso Típico:

Este hook se puede integrar en componentes React que necesiten manejar la lógica de visualización y edición de los datos de los comerciantes. Proporciona una interfaz clara para gestionar estados relacionados con pestañas, datos del comerciante y selecciones, mejorando la modularidad y reutilización del código.

4.2.2 Carpeta middleware

La carpeta "middleware" en la aplicación contiene funciones middleware cruciales para gestionar aspectos como la autorización, la seguridad y otros procesos intermedios. Estos middlewares actúan como una capa intermedia entre las solicitudes y respuestas, asegurando que ciertas condiciones o validaciones se cumplan antes de proceder con la ejecución normal de una operación.

AuthMiddleware:

En el contexto de esta carpeta, destaca la función "AuthMiddleware," diseñada para manejar procesos de autenticación. Su propósito principal es garantizar que el usuario esté autenticado antes de permitir el acceso a ciertas partes de la aplicación. En caso de que el usuario no esté autenticado, se le redirige a la ruta principal ("/") para mantener la seguridad y la coherencia en el flujo de la aplicación.

Explicación Detallada:

- **Parámetros:**
  - user: Representa el estado de autenticación del usuario. Si es null, undefined, o cualquier evaluación "falsy," se considera que el usuario no está autenticado.
  - navigate: Función comúnmente utilizada para la navegación en una aplicación React Router. En caso de que el usuario no esté autenticado, se utiliza la función navigate para redirigir al usuario a la página de inicio ('/').
- **Uso Típico:**
  - Este middleware se implementa en puntos específicos de la aplicación para garantizar que solo los usuarios autenticados tengan acceso a ciertas partes de la misma. Aquí hay un ejemplo de la aplicación:
  - Se puede utilizar en rutas protegidas para evitar que usuarios no autenticados accedan a páginas que requieren autenticación.
  - Este middleware se integra en el flujo de la aplicación donde la autenticación es crucial, proporcionando una capa de seguridad en esas secciones específicas.

Al utilizar este middleware de manera efectiva, se contribuye a un sistema más seguro y coherente, asegurando que las partes sensibles de la aplicación solo estén disponibles para usuarios debidamente autenticados

5. Componentes Reutilizables

5.1 BarChart

Este componente, llamado BarChart, se encarga de renderizar un gráfico de barras utilizando la librería Chart.js en una aplicación React. Aquí tienes una explicación detallada de sus características y funcionalidades:

Dependencias Importadas:

- React: Necesaria para la creación de componentes de React.
- useEffect y useRef de React: Utilizados para gestionar efectos secundarios y mantener una referencia mutable.

Propiedades (Props):

- data: Contiene la información necesaria para construir el gráfico de barras. Esta información incluye las etiquetas y los conjuntos de datos.
- chartId: Un identificador único para el gráfico. Se utiliza para acceder al elemento canvas asociado en el DOM.

Estructura y Funciones Principales:

- BarChart Function Component:
  - Este componente funcional retorna un elemento canvas con un ancho flexible y una altura fija de 190 píxeles.
  - La referencia (chartRef) se utiliza para rastrear el objeto de gráfico Chart.js y realizar operaciones en él.
- useEffect Hook:
  - Este hook se utiliza para realizar operaciones secundarias después de que el componente ha sido renderizado.
  - Se encarga de inicializar o actualizar el gráfico de barras cuando las propiedades data o chartId cambian.

- Garantiza la limpieza adecuada del gráfico anterior antes de crear uno nuevo.

- Creación del Gráfico de Barras:
  - Se accede al elemento canvas del DOM a través de la referencia y se destruye cualquier gráfico previo para evitar conflictos.
  - Se definen las opciones del gráfico, como la escala del eje y, para personalizar su apariencia.
  - Se crea un nuevo gráfico de barras utilizando la librería Chart.js, pasando los datos y opciones configuradas.
- Limpieza del Gráfico:
  - Se retorna una función de limpieza desde el useEffect, asegurando que el gráfico anterior se destruya al desmontar el componente.

Uso del Componente:

- Este componente puede ser utilizado en otras partes de la aplicación donde se desee mostrar un gráfico de barras.
- Se proporciona con datos específicos (data) y un identificador único (chartId) para personalizar su contenido y apariencia.

En resumen, el componente BarChart encapsula la lógica necesaria para la creación y actualización de un gráfico de barras interactivo y dinámico en una aplicación React, brindando una representación visual eficaz de datos estadísticos.

5.2 CheckboxInput

El componente CheckboxInput es una implementación de un cuadro de entrada (input) con funcionalidad de lista desplegable que permite seleccionar días de la semana. A continuación, se proporciona una explicación detallada:

Dependencias Importadas:

- React y useState: Utilizados para crear componentes y gestionar el estado local.

Propiedades (Props):

- selectedDays: Un array que contiene los días seleccionados actualmente.

- setSelectedDays: Una función para actualizar el estado de los días seleccionados.

Estado Local:

- isOpen: Un estado que rastrea si la lista desplegable está abierta o cerrada.

Funciones Principales:

- toggleDropdown Function: Cambia el estado isOpen cuando se hace clic en el cuadro de entrada o en el ícono de calendario.
- handleDayToggle Function: Agrega o quita un día del array de días seleccionados según su estado actual.
- placeholderText Variable: Determina el texto del marcador de posición del cuadro de entrada según los días seleccionados.

Estructura del Componente:

- El componente consiste en un contenedor que contiene un cuadro de entrada y un ícono de calendario.
- Cuando se hace clic en el cuadro de entrada o en el ícono de calendario, se muestra una lista desplegable de días de la semana.

Componente Interno: CheckboxOption

- Representa una opción de checkbox para un día específico de la semana.
- Contiene un checkbox que se marca o desmarca según si el día está seleccionado.
- Muestra el ícono de verificación (FiCheck) en caso de que el día esté seleccionado.
- La etiqueta del día se muestra junto al checkbox.

Uso del Componente:

- Se puede incorporar en otras partes de la aplicación donde se necesite una interfaz para seleccionar días de la semana.

5.3 CheckboxOption

El componente CheckboxOption es un subcomponente de CheckboxInput y representa una opción de checkbox para un día específico de la semana. A continuación, se proporciona una explicación detallada:

Propiedades (Props):

- day: Una abreviatura del día de la semana (por ejemplo, "Lun" para lunes).
- label: El nombre completo del día de la semana.
- handleToggle: Una función para manejar el cambio de estado del checkbox.
- selected: Un valor booleano que indica si el día está actualmente seleccionado.

Estructura y Funciones Principales:

- Checkbox y FiCheck: El componente utiliza un checkbox de HTML estándar y muestra el ícono de verificación (FiCheck) solo si el día está seleccionado.
- Etiqueta del Día: Muestra la etiqueta del día de la semana junto al checkbox.
- Manejo de Cambios: Cuando se cambia el estado del checkbox, la función handleToggle se llama para actualizar el estado en el componente principal (CheckboxInput).

Uso del Componente:

- Este componente se utiliza internamente en CheckboxInput para representar cada día de la semana con su correspondiente opción de checkbox.

En resumen, CheckboxInput y CheckboxOption forman un componente completo que permite seleccionar días de la semana a través de un cuadro de entrada con lista desplegable y opciones de checkbox interactivas.

5.4 ConceptList

El componente ConceptList es una representación de una fila de una lista de conceptos en una tabla. A continuación, se proporciona una explicación detallada:

Propiedades (Props):

- importe: El importe asociado al concepto.

- unidad: La unidad de medida del concepto.
- metraje: El metraje asociado al concepto.
- totalDaysWorked: El total de días trabajados asociados al concepto.

Estructura del Componente:

- El componente representa una fila (tr) en una tabla.
- Contiene seis celdas (td) que representan diferentes aspectos del concepto.

Contenido de las Celdas:

- Celda 1 (importe): Muestra el importe del concepto precedido por el símbolo de dólar.
- Celda 2: Muestra el número "1", posiblemente indicando una cantidad.
- Celda 3 (metraje): Muestra el metraje asociado al concepto.
- Celda 4 (totalDaysWorked): Muestra el total de días trabajados asociados al concepto.
- Celda 5 (importe): Similar a la Celda 1, muestra el importe del concepto precedido por el símbolo de dólar.
- Celda 6 (unidad): Muestra la unidad de medida del concepto en minúsculas.

Estilos y Clases:

- Se aplican estilos específicos utilizando clases de Tailwind CSS (w-2/12, w-1/12, etc.) para establecer el ancho de cada celda en proporción.

Notas Adicionales:

- La fórmula utilizada para mostrar el importe ('$' + importe + '.00') sugiere que se trata de valores de moneda con dos decimales fijos.

Uso del Componente:

- Este componente puede ser utilizado en una tabla para representar un concepto particular con detalles asociados.

En resumen, ConceptList es un componente que presenta información relacionada con un concepto en forma de fila de tabla, proporcionando una representación clara y estructurada de los detalles del concepto.

5.5 CountsDayBetweenDates

El componente countsDayBetweenDates incluye una función llamada contarDiasDeLaSemana que calcula la cantidad de días de una semana específica entre dos fechas dadas. A continuación, se proporciona una explicación detallada:

Función: contarDiasDeLaSemana:

- Parámetros:
  - fechaInicialSinFormato: Fecha de inicio en formato no procesado.
  - fechaFinalSinFormato: Fecha de finalización en formato no procesado.
  - diaDeLaSemana: Día de la semana que se está contando.
- Proceso:
  - La función utiliza la función auxiliar formatearfechas para formatear las fechas de inicio y finalización.
  - Inicia un bucle que recorre todos los días desde la fecha de inicio hasta la fecha de finalización.
  - En cada iteración, verifica si el día de la semana actual coincide con el día especificado (diaDeLaSemana) utilizando un diccionario (diccionarioDaysOfWeek) que asigna nombres de días a números de día de la semana (0 a 6).
  - Incrementa un contador si la condición se cumple.
  - Devuelve el contador que representa la cantidad de días de la semana especificada entre las fechas dadas.

Función auxiliar: formatearfechas:

- Parámetro:
  - fechaSinFormato: Fecha en formato no procesado.
- Proceso:
  - Convierte la fecha sin formato en un objeto de fecha.
  - Extrae el año, mes y día de la fecha.
  - Formatea la fecha en el formato "YYYY-MM-DD".
  - Devuelve la fecha formateada.

Diccionario de Días de la Semana:

- Se crea un diccionario (diccionarioDaysOfWeek) que asigna nombres de días a números de día de la semana (0 a 6).

Uso del Componente:

- Esta función puede ser utilizada en situaciones donde se necesita contar la cantidad de días de una semana específica entre dos fechas, proporcionando flexibilidad para adaptarse a diversos escenarios de aplicación.

En resumen, countsDayBetweenDates es un componente que ofrece una función para calcular la cantidad de días de una semana específica entre dos fechas, contribuyendo a la lógica de manejo de fechas en la aplicación.

5.6 DatePickerInput

El componente DatePickerInput es una envoltura que utiliza la biblioteca react-datepicker para proporcionar una interfaz de selección de fechas. A continuación, se presenta una explicación detallada:

Propiedades (Props):

- setSelectedDate: Función para actualizar el estado del componente padre con la fecha seleccionada.
- selectedDate: Fecha seleccionada actual.

Función: handleDateChange:

- Parámetro:
  - date: Nueva fecha seleccionada.
- Proceso:
  - Llama a setSelectedDate para actualizar el estado con la nueva fecha seleccionada.

Estructura del Componente:

- Utiliza el componente DatePicker de la biblioteca react-datepicker para manejar la selección de fechas.

- Contiene un ícono de calendario (FiCalendar) que indica visualmente la función del componente.

Estilo y Clases:

- Se aplican estilos específicos utilizando clases de Tailwind CSS (relative, inline-flex, rounded-lg, etc.) para proporcionar una apariencia específica al componente.

Uso del Componente:

- Este componente puede ser utilizado en formularios u otras secciones donde se requiera que los usuarios seleccionen una fecha.

En resumen, DatePickerInput es un componente que encapsula la funcionalidad de selección de fechas proporcionada por react-datepicker. Ofrece una interfaz sencilla y estilizada para que los usuarios elijan fechas, con la capacidad de actualizar el estado del componente padre con la fecha seleccionada.

5.7 DisableableButton

El componente DisableableButton es un botón que puede deshabilitarse basándose en ciertas condiciones. Aquí se presenta una explicación detallada:

Propiedades (Props):

- selectedOption: Opción seleccionada, posiblemente utilizada para validar si está seleccionada.
- selectBeginDate: Función para seleccionar la fecha de inicio.
- selectEndDate: Función para seleccionar la fecha de fin.
- selectedDays: Días seleccionados, posiblemente utilizados para validar si hay días seleccionados.
- agregaConceptoPago: Función que se ejecutará al hacer clic en el botón.

Condiciones de Deshabilitación:

- noOptionSelected: No hay opción seleccionada.
- optionIsNotPesos: La opción seleccionada no es en pesos y falta alguna de las fechas o no hay días seleccionados.

- missingBeginDate: Falta la fecha de inicio.
- missingEndDate: Falta la fecha de fin.
- noSelectedDays: No hay días seleccionados.

Deshabilitación del Botón:

- El botón se deshabilita si cualquiera de las condiciones de deshabilitación es verdadera.

Estilo y Clases:

- Se aplican estilos específicos utilizando clases de Tailwind CSS (w-full, h-9, font-Foco-Corp-Bold, etc.) para proporcionar un estilo específico al botón.
- El color y la opacidad del botón pueden cambiar según si está deshabilitado o no.

Manejo de Eventos:

- Al hacer clic en el botón, se ejecuta la función agregaConceptoPago si el botón no está deshabilitado.

Uso del Componente:

- Este componente puede ser utilizado en formularios u otras secciones donde se necesite un botón que pueda deshabilitarse según ciertas condiciones.

En resumen, DisableableButton es un componente que representa un botón con la capacidad de deshabilitarse en función de diversas condiciones, brindando una experiencia de usuario más controlada.

5.8 FormatDates

El archivo formatDates.js contiene dos funciones para formatear fechas de diferentes maneras. A continuación, se ofrece una descripción detallada:

Funciones:

setDateFormat

- Descripción: Convierte una fecha en formato "dd de mes de año".
- Parámetros:
- date (tipo: Date): La fecha a formatear.

- Retorna (tipo: string): La fecha formateada.

setDateFormatDDMMYY

- Descripción: Convierte una fecha en formato "dd/MM/yy".
- Parámetros:
- date (tipo: Date): La fecha a formatear.
- Retorna (tipo: string): La fecha formateada.

Ambas funciones utilizan el objeto Date de JavaScript y métodos relacionados para obtener y formatear los componentes de la fecha, como día, mes y año.

Uso de las Funciones:

Estas funciones pueden ser importadas y utilizadas en otros componentes o módulos que necesiten formatear fechas de acuerdo con los patrones especificados. Por ejemplo, para mostrar fechas en la interfaz de usuario de una manera más amigable o específica.

5.9 Header

El componente Header es una barra de encabezado que se utiliza para mostrar el logo de la empresa y la navegación o título correspondiente a la página actual. Lo siguiente es una descripción detallada:

Propiedades (Props):

- useButton: Un booleano que determina si se muestra un botón en lugar del título.
- currentPage: La página actual, utilizado para determinar el contenido dinámico del encabezado.

Funciones:

- cerrarSesion: Una función que se ejecuta cuando se hace clic en el botón "Cerrar sesión", redirigiendo al usuario a la página de inicio.

Estructura del Componente:

- Contiene un contenedor header que alberga el logo de la empresa y el título o botón, dependiendo del contexto.

- Utiliza un img para mostrar el logo de la empresa.
- Muestra un título específico o un botón "Cerrar sesión" según la página actual.

Componentes Adicionales Utilizados:

- DashBoard_Ejecutivo: Un componente de vista correspondiente al dashboard ejecutivo.

Contenido Dinámico:

- Si useButton es verdadero, muestra un botón que ejecuta la función cerrarSesion. El título del botón es "Cerrar sesión" y se acompaña de un icono.
- Si useButton es falso, muestra el título específico de la página actual.

Barra de Colores:

- Abajo del encabezado, presenta una línea de colores con segmentos de diferentes colores.

Uso del Componente:

- Este componente puede ser utilizado como una barra de encabezado en diversas páginas, ajustando dinámicamente su contenido según la página actual.
- Puede mostrar un botón "Cerrar sesión" en el dashboard ejecutivo o un título específico en otras páginas.

En resumen, Header es un componente versátil que proporciona una interfaz de usuario coherente con la capacidad de adaptarse a diferentes contextos y necesidades de navegación.

5.10 InfoComponent

El componente InfoComponent es una tarjeta informativa que muestra detalles específicos sobre un comerciante o comercio. A continuación, se presenta una descripción detallada:

Propiedades (Props):

- folio: El número de folio asociado al comerciante o comercio.
- nombre: El nombre del comerciante o comercio.
- giroActivo: El giro o actividad económica del comerciante o comercio.
- observaciones: Información adicional u observaciones sobre el comerciante o comercio.
- fecha_termino: La fecha de terminación o vigencia de la información.
- tercera_edad: Un indicador booleano que representa si el comerciante o comercio pertenece a la tercera edad.

Funciones:

- verTodo: Una función que se ejecuta cuando se hace clic en el botón "VER TODO", redirigiendo al usuario a una página detallada del comerciante o comercio.

Estructura del Componente:

- Utiliza un contenedor div con clases dinámicas de colores (colorAll) que representan el estado del comerciante o comercio (rojo para observaciones, azul para tercera edad, verde para vigente, gris para otros casos).
- Incluye una sección superior con el número de folio.
- Muestra información detallada como nombre, giro/activo, y observaciones.
- Presenta la fecha de vigencia en la sección inferior.
- Incluye un botón "VER TODO" que ejecuta la función verTodo.

Estilos y Clases:

- Aplica clases de Tailwind CSS para establecer el diseño y colores de la tarjeta.

Contenido Dinámico:

- El color y el contenido de la tarjeta varían según las propiedades proporcionadas (observaciones, tercera_edad, fecha_termino).

Uso del Componente:

- Este componente puede ser utilizado en una lista de comerciantes o como una tarjeta individual en una interfaz de usuario que presenta información resumida.

En resumen, InfoComponent es un componente versátil que proporciona una vista resumida y colorida de la información clave sobre un comerciante o comercio, con la capacidad de navegar a detalles más extensos al hacer clic en el botón "VER TODO".

5.11 Middleware

El componente Middleware contiene funciones y componentes que se utilizan para gestionar la protección de rutas y la funcionalidad de cierre de sesión en la aplicación. Aquí hay una descripción detallada:

Funciones:

ExecutiveProtectedRoute

- Propósito: Este componente proporciona una ruta protegida para usuarios ejecutivos.
- Propiedades (Props): children - Componentes secundarios que deben renderizarse dentro del componente.
- Funcionalidad:
  - Comprueba el tipo de usuario almacenado en el almacenamiento local.
  - Si el usuario no es ejecutivo (userBoolean no es false), redirige al usuario a la ruta principal (/).
  - Si el usuario es ejecutivo, renderiza los componentes secundarios.

LogOut

- Propósito: Este componente se utiliza para cerrar sesión.
- Propiedades (Props): setuserData - Función para actualizar los datos del usuario.
- Funcionalidad:

- Utiliza useEffect para realizar la operación de cierre de sesión cuando el componente se monta.
- Limpia el almacenamiento local.

AdminProtectedRoute

- Propósito: Este componente proporciona una ruta protegida para usuarios administradores.
- Propiedades (Props): children - Componentes secundarios que deben renderizarse dentro del componente.
- Funcionalidad:
  - Comprueba el tipo de usuario almacenado en el almacenamiento local.
  - Si el usuario no es administrador (userBoolean no es true), redirige al usuario a la ruta principal (/).
  - Si el usuario es administrador, renderiza los componentes secundarios.

Uso del Componente:

- Estos componentes pueden ser utilizados como contenedores de rutas en la aplicación para garantizar que ciertas rutas estén protegidas y que el cierre de sesión se maneje adecuadamente.

En resumen, el componente Middleware proporciona funciones y componentes esenciales para gestionar la protección de rutas y la funcionalidad de cierre de sesión en la aplicación, asegurando un flujo de navegación seguro y controlado.

5.12 Modal

El componente Modal es un cuadro de diálogo modal que permite al usuario refrendar un comercio. Aquí hay una descripción detallada:

Propiedades (Props):

- closeModal: Una función para cerrar el modal.

Estado:

- pendding: Un estado de React que indica si la operación de refrendar está pendiente o en curso.

Funciones:

- handleRefrendarClick: Una función que se ejecuta cuando se hace clic en el botón "Refrendar". Realiza una simulación de una llamada a la API y maneja diferentes casos según la respuesta.
- openCedula: Una función que abre una nueva ventana con la cédula del comercio.
- reloadWindow: Una función que recarga la ventana después de refrendar un comercio.
- otherCases: Una función que maneja otros casos, como la registración de una orden de pago.
- showErrorAlert: Una función que muestra una alerta de error en caso de fallo en la operación.

Estructura del Componente:

- Utiliza un contenedor div para el modal con estilos específicos de Tailwind CSS.
- Contiene un formulario con campos para la referencia de pago.
- Incluye botones para refrendar o cancelar la operación.
- Utiliza SweetAlert2 para mostrar alertas visuales al usuario.
- Muestra un spinner de carga durante la operación de refrendar.

Estilos y Clases:

- Aplica clases de Tailwind CSS para establecer el diseño y colores del modal y sus elementos.

Contenido Dinámico:

- El botón de refrendar muestra "Refrendar" o un spinner de carga según el estado pendding.
- Las alertas visuales varían según la respuesta de la llamada a la API.

Uso del Componente:

- Este componente puede ser utilizado como un cuadro de diálogo modal para refrendar comercios.

En resumen, Modal es un componente interactivo que permite al usuario refrendar un comercio y muestra alertas visuales informativas basadas en la respuesta de la llamada a la API.

5.13 Sidebar

El componente Sidebar representa una barra lateral de navegación en la interfaz de usuario. Aquí hay una descripción detallada:

Propiedades (Props):

- isOpen: Un booleano que indica si la barra lateral está abierta o cerrada.

Estructura del Componente:

- Utiliza un contenedor div con estilos específicos de Tailwind CSS para la barra lateral.
- Contiene enlaces (Link de React Router) para diferentes secciones, como "Inicio", "Ingresos" y "Comercios".
- Incluye un enlace para cerrar sesión.
- Utiliza iconos de React para representar visualmente cada sección y el botón de cerrar sesión.

Estilos y Clases:

- Aplica clases de Tailwind CSS para establecer el diseño, colores y transiciones de la barra lateral.
- Utiliza transiciones para animar la apertura y cierre de la barra lateral.

Contenido Dinámico:

- Los enlaces y el botón de cerrar sesión cambian su apariencia en función de si el cursor está sobre ellos.

Uso del Componente:

Este componente puede ser utilizado como una barra lateral de navegación en aplicaciones web para proporcionar accesos rápidos a diferentes secciones.

Funcionalidad:

- Los enlaces redirigen a diferentes rutas dentro de la aplicación.

- El botón de cerrar sesión podría llevar al usuario de vuelta a la página de inicio de sesión.

En resumen, Sidebar es un componente funcional y visualmente atractivo que facilita la navegación en la interfaz de usuario de la aplicación web.

5.14 SelectStyles

El componente SelectStyles proporciona un conjunto de estilos predefinidos para personalizar la apariencia y el comportamiento de un componente de selección (select) en una interfaz de usuario. Aquí hay una descripción detallada:

Estilos del Componente:

1. **control**
   - Propósito: Establece los estilos para el contenedor principal del componente de selección.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.
     - state: Estado actual del componente de selección.
   - Funcionalidad:
     - Cambia el color de fondo al hacer foco (state.isFocused) a blanco (#FFF).
     - Define la altura del componente (height: '36px').
     - Configura el borde, la esquina redondeada y la sombra del componente.
     - Cambia el color del borde al hacer hover (&:hover) a naranja (#F18A00).
2. **option**
   - Propósito: Establece los estilos para cada opción dentro del componente de selección.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.
     - state: Estado actual de la opción.
   - Funcionalidad:
     - Cambia el color de fondo si la opción está seleccionada (state.isSelected) o si está enfocada (state.isFocused) a naranja (#F18A00).

- Cambia el color del texto a blanco si la opción está seleccionada o enfocada.

3. **dropdownIndicator**
   - Propósito: Establece los estilos para el indicador de despliegue del menú del componente de selección.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.
   - Funcionalidad:
     - Define el relleno (padding) del indicador.
     - Establece esquinas redondeadas en la parte superior derecha e inferior derecha (borderTopRightRadius y borderBottomRightRadius).

Uso del Componente:

- Estos estilos pueden ser utilizados como una constante en otros componentes que utilicen la biblioteca de selección (por ejemplo, react-select) para personalizar la apariencia del componente de selección.

En resumen, SelectStyles es una constante que proporciona estilos personalizados para mejorar la apariencia y experiencia del usuario al interactuar con un componente de selección en una aplicación.

5.15 SelectStylesForm

El componente SelectStylesForm proporciona un conjunto de estilos específicos para personalizar la apariencia y el comportamiento de un componente de selección (select) en formularios de una interfaz de usuario. Aquí hay una descripción detallada:

Estilos del Componente:

1. **control**
   - Propósito: Establece los estilos para el contenedor principal del componente de selección en formularios.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.

- state: Estado actual del componente de selección.

- Funcionalidad:
  - Cambia el color de fondo al hacer foco (state.isFocused) a un tono más claro (#F9FAFB).
  - Define la esquina redondeada del componente (borderRadius: '8px').
  - Configura el borde del componente en función del foco y el estado normal.
  - Elimina la sombra del componente.
  - Ajusta la altura mínima del componente (minHeight: '100% !important').
  - Cambia el color del borde al hacer hover (&:hover) a naranja (#F18A00).

2. **option**
   - Propósito: Establece los estilos para cada opción dentro del componente de selección en formularios.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.
     - state: Estado actual de la opción.
   - Funcionalidad:
     - Cambia el color de fondo si la opción está seleccionada (state.isSelected) o si está enfocada (state.isFocused) a naranja (#F18A00).
     - Cambia el color del texto a blanco si la opción está seleccionada o enfocada.
3. **dropdownIndicator**
   - Propósito: Establece los estilos para el indicador de despliegue del menú del componente de selección en formularios.
   - Propiedades:
     - provided: Estilos proporcionados por defecto.
   - Funcionalidad:
     - Define el relleno (padding) del indicador.
     - Establece esquinas redondeadas en la parte superior derecha e inferior derecha (borderTopRightRadius y borderBottomRightRadius).

Uso del Componente:

- Estos estilos están diseñados específicamente para formularios, proporcionando una apariencia y comportamiento mejorados para la interacción del usuario con el componente de selección.

En resumen, SelectStylesForm es una constante que ofrece estilos personalizados para mejorar la experiencia del usuario al interactuar con un componente de selección en formularios dentro de una aplicación.

5.16 SwtAlerts

Este archivo es un conjunto de funciones utiliza la biblioteca sweetalert2 para mostrar alertas modales en una aplicación. Cada función está diseñada para presentar un tipo específico de alerta con un ícono y contenido personalizados. Aquí tienes una explicación detallada:

showInfoAlert Function:

- **Descripción:** Esta función muestra una alerta modal informativa con un ícono de información.
- **Parámetros:**
  - title: El título de la alerta.
  - text: El contenido o mensaje de la alerta.

showInfoAlertOp Function:

- **Descripción:** Similar a showInfoAlert, esta función también muestra una alerta modal informativa con un ícono de información. Es posible que esta función esté duplicada accidentalmente, ya que parece tener la misma implementación que showInfoAlert.
- **Parámetros:**
  - title: El título de la alerta.
  - text: El contenido o mensaje de la alerta.

showErrorAlert Function:

- **Descripción:** Esta función muestra una alerta modal de error con un ícono de error.

- **Parámetros:**
  - title: El título de la alerta.
  - text: El contenido o mensaje de la alerta.

Estas funciones utilizan la biblioteca sweetalert2 para crear y mostrar las alertas, especificando el tipo de ícono (info o error), el título y el texto que se deben mostrar en la alerta. Por ejemplo, puedes llamar a showInfoAlert("Título", "Este es un mensaje informativo") para mostrar una alerta informativa.

5.17 TabButton

El componente TabButton representa un botón de pestaña en la interfaz de usuario. Aquí hay una descripción detallada:

Propiedades (Props):

- label: Un objeto que contiene información sobre la pestaña, como el recuento y el texto.
- isActive: Un booleano que indica si la pestaña está activa o no.
- onClick: Una función que se ejecuta al hacer clic en la pestaña.

Estructura del Componente:

- Utiliza un contenedor div con estilos dinámicos basados en las propiedades recibidas.
- Contiene elementos HTML y estilos específicos de Tailwind CSS para establecer el diseño y la apariencia del botón.

Estilos y Clases:

- Aplica clases de Tailwind CSS y estilos condicionales para cambiar la apariencia del botón según si está activo o inactivo.
- Utiliza estilos opacos y sombras para darle profundidad visual al botón.

Contenido Dinámico:

- El contenido del botón varía en función de las propiedades recibidas. Muestra el recuento y el texto de la pestaña.

Uso del Componente:

- Este componente puede ser utilizado como un botón de pestaña en interfaces de usuario que necesitan cambiar entre diferentes vistas o secciones.

Funcionalidad:

- La función onClick se ejecuta al hacer clic en el botón, lo que permite manejar acciones específicas asociadas a la pestaña.
- Cambia dinámicamente su apariencia según si está activo o inactivo.

En resumen, TabButton es un componente visualmente atractivo y funcional que facilita la representación de pestañas en la interfaz de usuario. Su diseño modular y la capacidad de personalizar su apariencia lo hacen útil en diversas aplicaciones.

5.18 Table

El componente Table representa una tabla en la interfaz de usuario. Aquí hay una descripción detallada:

Propiedades (Props):

- data: Una matriz que contiene los datos que se mostrarán en la tabla.
- headers: Una matriz que contiene los encabezados de las columnas de la tabla.
- numColumns: Un número que especifica el número de columnas a mostrar.

Estructura del Componente:

- Utiliza un contenedor div con la clase overflow-auto para permitir el desplazamiento vertical si la tabla es más grande que el contenedor.
- Contiene un elemento table con estilos específicos de Tailwind CSS.

Elementos:

- La tabla tiene una sección thead que contiene una fila de encabezado con estilos específicos.
- La fila de encabezado contiene celdas th para cada encabezado de columna.
- La tabla tiene una sección tbody que contiene filas y celdas con datos.
- Las celdas de datos (td) muestran los datos correspondientes.

Estilos y Clases:

- Se aplican clases de Tailwind CSS para establecer el diseño, colores y estilo de borde de la tabla y las celdas.
- Se utiliza la clase sticky para mantener los encabezados visibles mientras se desplaza verticalmente por la tabla.

Contenido Dinámico:

- Los encabezados y datos se generan dinámicamente en función de las propiedades recibidas.

Uso del Componente:

- Este componente puede ser utilizado para mostrar datos en formato de tabla en diversas partes de la interfaz de usuario.

En resumen, Table es un componente funcional y versátil que facilita la representación de datos en forma de tabla en la interfaz de usuario. Su diseño modular y la capacidad de personalizar la apariencia lo hacen útil en diversas aplicaciones.

6. Vistas de la Aplicación

La aplicación está compuesta por varias vistas que están organizadas en carpetas específicas según su funcionalidad. A continuación, se detalla cada una de las vistas presentes en las carpetas admin, executive, y PDF, así como el archivo de inicio de sesión login.

6.1 Archivo Login:

Descripción:
El componente Login maneja la autenticación de usuarios y presenta un formulario de inicio de sesión. Se encarga de interactuar con la API para verificar las credenciales y redirigir a las vistas correspondientes según el tipo de usuario.

Importaciones:

- React, { useState, useEffect, useRef }: Importa React y los hooks useState, useEffect, useRef.
- axios: Importa la biblioteca Axios para realizar solicitudes HTTP.
- showInfoAlert, showErrorAlert: Importa funciones para mostrar alertas.
- useNavigate: Importa el hook useNavigate de react-router-dom.

Estado del Componente:

- Utiliza hooks de estado (useState) para gestionar el estado del nombre de usuario (username) y la contraseña (password).
- Utiliza el hook useRef para referenciar los campos de entrada de usuario y contraseña.
- Inicializa el estado de userData con el prop recibido.

Funciones:

- useEffect: Se ejecuta al renderizar el componente para limpiar el almacenamiento local y enfocarse en el campo de usuario.
- onChangeUsername: Maneja los cambios en el campo de usuario y actualiza el estado correspondiente.
- onChangePassword: Maneja los cambios en el campo de contraseña y actualiza el estado correspondiente.
- handleKeyDown: Maneja las pulsaciones de teclas, permitiendo la transición entre campos y la acción de inicio de sesión al presionar Enter.
- handleLogin: Realiza la solicitud de inicio de sesión a la API. Almacena el tipo de usuario en el almacenamiento local y redirige a las vistas correspondientes.

Renderizado JSX:

- Estructura general: Incluye una sección de imágenes/logo, seguida de un formulario de inicio de sesión.
- Campos de entrada: Dos campos de entrada para el nombre de usuario y la contraseña, con referencias y manejo de eventos.

- Botón de inicio de sesión: Un botón que activa la función handleLogin al hacer clic.
- Utiliza clases de estilo de Tailwind CSS para el diseño y estilos de los elementos.

Este componente proporciona una interfaz de inicio de sesión amigable y maneja la lógica de autenticación de manera eficiente. Además, utiliza alertas para informar al usuario sobre posibles errores o mensajes importantes.

6.2 AdminViews:

6.2.1 DashBoard_Admin:

Descripción:

La vista del "Dashboard del Administrador" proporciona una visión general de las estadísticas clave y la información relevante para el administrador del sistema. A continuación, se proporciona una descripción detallada de la vista:

Importaciones:

- React, { useEffect, useState }: Importa React y los hooks useEffect y useState.
- AuthMiddleware: Importa un middleware de autenticación (no proporcionado en el código compartido).
- useLocation, useNavigate: Importa hooks de react-router-dom para manejar la ubicación y navegación en la aplicación.
- Header, Sidebar, BarChart: Importa componentes de la aplicación.
- IoGrid, IoPerson, IoChevronForwardCircle, IoPeopleSharp, IoStorefrontSharp, GiCash: Importa iconos de la librería react-icons/io5 y react-icons/gi.
- axios: Importa la biblioteca Axios para realizar solicitudes HTTP.

Función DashBoard_Admin:

- Define el componente funcional DashBoard_Admin.

Estado del Componente:

- Utiliza hooks de estado (useState) para gestionar el estado del panel lateral, los datos y gráficos
  - isOpen: Estado para controlar si el panel lateral está abierto o cerrado.
  - data: Estado para almacenar los datos del dashboard.
  - dayGraph, weekGraph, monthGraph: Estados para almacenar datos específicos de los gráficos diarios, semanales y mensuales respectivamente.
- Utiliza el hook useEffect para realizar una solicitud HTTP y cargar los datos del dashboard cuando el componente se monta.
  - Hace una solicitud a la URL especificada para obtener información del dashboard.
  - Los datos se almacenan en los estados correspondientes (data, dayGraph, weekGraph, monthGraph).
- Define funciones como toggleSidebar para manejar la apertura/cierre del panel lateral.
  - Cambia el estado isOpen para alternar entre, abrir y cerrar el panel lateral.

Datos y Gráficos:

- dataChart1, dataChart2, y dataChart3 contienen la configuración de datos para los gráficos de barras.
- Los datos para los gráficos se extraen de los estados dayGraph, weekGraph, y monthGraph.

Renderizado JSX:

- Muestra el componente Header.
- Utiliza un diseño de dos columnas (flex flex-row).
- Renderiza el componente Sidebar con la capacidad de ser mostrado u ocultado.
- Contiene varios contenedores div que representan diferentes secciones del panel de administrador.

- Utiliza clases de estilo basadas en Tailwind CSS para el diseño y estilos de los elementos.
- Renderiza varios bloques de estadísticas con iconos y botones para ver más detalles.
- Muestra tres gráficos de barras usando el componente BarChart.
- Incluye un botón para ver todos los ingresos.

Estas son las principales funciones y variables dentro del componente DashBoard_Admin. Cada función y variable cumple un propósito específico para manejar el estado, la lógica y la presentación del componente.

6.2.2 IncomeView:

Descripción:

La vista IncomeView se encarga de mostrar información detallada sobre los ingresos, incluyendo tablas y gráficos que resumen los ingresos diarios, semanales, mensuales y anuales. También permite filtrar la información por fechas específicas.

Importaciones:

- React, { useEffect, useState }: Importa React y los hooks useEffect y useState.
- Header, Sidebar, TabButton, Table, BarChart, DatePickerInput: Importa componentes de la aplicación.
- IoGrid, IoArrowBack: Importa iconos de la librería react-icons/io5.
- axios: Importa la biblioteca Axios para realizar solicitudes HTTP.

Estado del Componente:

- Utiliza hooks de estado (useState) para gestionar el estado del panel lateral (isOpen) y los datos generales (data).
- Utiliza el hook useEffect para realizar una solicitud HTTP y cargar los datos de ingresos cuando el componente se monta.
  - Hace una solicitud a la URL especificada para obtener información de ingresos.
  - Los datos se almacenan en el estado correspondiente (data).

- Otros estados incluyen activeTab para gestionar la pestaña activa y datos específicos de tablas (table1Data, table2Data, table3Data, table4Data) y gráficos (dataChartDay, dataChartWeek, dataChartMonth, dataChartYear).

Funciones:

- toggleSidebar: Cambia el estado isOpen para alternar entre, abrir y cerrar el panel lateral.
- handleTabClick: Actualiza el estado activeTab al hacer clic en una pestaña.

Renderizado JSX:

- Estructura general: Incluye el componente Header, Sidebar y una sección principal con información detallada sobre los ingresos.
- Sección de ingresos: Muestra información general sobre los ingresos y botones para cambiar entre pestañas (Día, Semana, Mes, Año).
- renderContent: Función que renderiza el contenido específico de cada pestaña, mostrando tablas y gráficos basados en los datos proporcionados.

Esta vista sigue un diseño modular y utiliza componentes reutilizables para mantener un código limpio y organizado. Además, implementa un diseño responsivo para adaptarse a diferentes tamaños de pantalla.

6.3 ExecutiveViews:

6.3.1 Dashboard_Executive:

Descripción:

Este componente representa el panel de control para el usuario ejecutivo. Permite buscar y filtrar comerciantes, así como realizar acciones como dar de alta a nuevos comerciantes. La interfaz muestra una serie de campos de entrada y opciones de filtrado.

Importaciones:

- axios: Importa la biblioteca Axios para realizar solicitudes HTTP.
- React, { useEffect, useState }: Importa React y los hooks useEffect, useState.
- AuthMiddleware: Middleware para la autenticación.
- useLocation, useNavigate: Importa hooks de react-router-dom.

- InfoComponent: Componente utilizado para mostrar información de los comerciantes.
- Header: Componente de encabezado.
- AiFillCaretDown: Icono de flecha hacia abajo de React-icons/Ai.

Estado del Componente:

- comerciantes: Estado que almacena la información de los comerciantes obtenida de la API.
- ComerciantesComponents: Estado que almacena componentes InfoComponent para representar a los comerciantes en la interfaz.
- Estados para los valores de los campos de búsqueda y filtrado (nameOrId, Mostrar, filtrarPor, colonia).

Funciones:

- useEffect:
  - Gestiona la autenticación del usuario mediante el middleware AuthMiddleware.
  - Actualiza el estado comerciantes con los datos proporcionados por la ubicación.
  - Llama a setInfoComponent para actualizar el estado ComerciantesComponents en función de comerciantes.
  - Realiza solicitudes a la API según los cambios en los campos de búsqueda y filtrado.
- setInfoComponent: Convierte la información de los comerciantes en componentes InfoComponent.
- HandleInputChanges: Maneja los cambios en los campos de entrada y llama a funciones específicas en consecuencia.
- Funciones onChange*: Actualizan los estados correspondientes según los cambios en los campos de búsqueda y filtrado.

Renderizado JSX:

- Se compone de un encabezado (Header) y una sección principal con campos de búsqueda, opciones de filtrado y botones de acción.

- Utiliza clases de estilo de Tailwind CSS para el diseño y estilos de los elementos.
- Incluye un bucle para renderizar los componentes InfoComponent en función de ComerciantesComponents.

Este componente proporciona una interfaz para que el usuario ejecutivo interactúe con la información de los comerciantes, realice búsquedas y tome acciones específicas.

6.3.2 DataComerciante:

Descripción:

Vista para visualizar y gestionar datos específicos de un comerciante. La vista DataComerciante está diseñada para la gestión integral de datos relacionados con un comerciante y su actividad comercial.

Importaciones:

Aquí se importan varias bibliotecas y componentes que se utilizan en el código. Algunas de las importaciones notables incluyen:

- React: La biblioteca principal para construir interfaces de usuario en React.
- useEffect y useState: Hooks de React para manejar efectos secundarios y estados, respectivamente.
- Header: Un componente que parece ser una barra de encabezado.
- RiUser3Fill y RiMenuFill de react-icons/ri: Iconos de usuario y menú.
- MdStore de react-icons/md: Icono de tienda.
- Select de react-select: Un componente de selección para crear menús desplegables.
- Swal de sweetalert2: Para mostrar ventanas modales de alerta.
- useNavigate y useParams de react-router-dom: Hooks para la navegación y la obtención de parámetros de la URL.
- Modal: Un componente de modal.
- axios: Biblioteca para hacer solicitudes HTTP.
- setDateFormatDDMMYY y useParamsDataComerciante: Funciones y un hook personalizado.

Estado y Variables:

- navigate: Un objeto de navegación proporcionado por el hook useNavigate de react-router-dom. Se utiliza para redirigir a otras páginas.
- folio: Un parámetro obtenido de la URL usando el hook useParams de react-router-dom. Representa el identificador único del comerciante.
- States relacionados con pestañas y menús:
  - activeTab: Identifica la pestaña activa en la interfaz.
  - setActiveTab: Función para actualizar el estado de activeTab.
  - isMenuOpenTab1 y similares**: Indican si el menú está abierto en una determinada pestaña.
  - setIsMenuOpenTab1 y similares**: Funciones para actualizar los estados de apertura de menús.
  - showButtonsTab1 y similares**: Indican si se deben mostrar botones en una pestaña específica.
  - setShowButtonsTab1 y similares**: Funciones para actualizar los estados de visualización de botones.
  - fieldsEditableTab1 y similares**: Indican si los campos son editables en una pestaña específica.
  - setFieldsEditableTab1 y similares**: Funciones para actualizar los estados de editabilidad de campos.

States relacionados con datos del comerciante:

- data: Almacena los datos del comerciante.
- setData: Función para actualizar el estado data.
- originalData: Almacena los datos originales del comerciante.
- setOriginalData: Función para actualizar el estado originalData.
- telefonos: Almacena los números de teléfono del comerciante.
- setTelefonos: Función para actualizar el estado telefonos.
- telefono1 y telefono2: Almacenan los números de teléfono individualmente.
- setTelefono1 y setTelefono2: Funciones para actualizar los estados de los números de teléfono.

States relacionados con fechas y opciones seleccionadas:

- fecha_inicio y fecha_termino: Almacenan las fechas de inicio y término del comerciante.
- setFecha_inicio y setFecha_termino: Funciones para actualizar los estados de fechas.
- selectedClasificacion, selectedHorario, y selectedTipo: Almacenan las opciones seleccionadas en ciertos menús desplegables.
- setSelectedClasificacion, setSelectedHorario, y setSelectedTipo: Funciones para actualizar los estados de opciones seleccionadas.

Funciones

- **getTodoInfo:**
  - Descripción: Realiza una solicitud HTTP GET para obtener toda la información del comerciante a través de la API.
  - Uso: Al cargar la vista por primera vez, se utiliza para recuperar y mostrar la información del comerciante.
- **makeGetApiCall:**
  - Descripción: Realiza una solicitud HTTP GET genérica.
  - Uso: Usado internamente por getTodoInfo para hacer llamadas GET a la API.
- **handleApiResponse:**
  - Descripción: Maneja la respuesta exitosa de la solicitud GET, actualizando el estado de la vista.
  - Uso: Se ejecuta después de recibir una respuesta exitosa de la API.
- **handleApiError:**
  - Descripción: Maneja los errores de la solicitud GET, mostrando los errores en la consola.
  - Uso: Se ejecuta en caso de un error en la solicitud GET.
- **setDataAndPhones:**
  - Descripción: Establece los estados relacionados con los datos del comerciante y los teléfonos.

- Uso: Utilizado después de recibir datos de la API para actualizar los estados.

- **setSelectedValues:**
  - Descripción: Configura los valores seleccionados para ciertos campos basados en la información recibida de la API.
  - Uso: Configura opciones seleccionadas en los menús desplegables.
- **setFechas:**
  - Descripción: Formatea y establece las fechas de inicio y fin.
  - Uso: Utilizado para presentar las fechas de manera legible.
- **updateComerciante:**
  - Descripción: Realiza una solicitud HTTP PUT para actualizar la información del comerciante y su comercio.
  - Uso: Se llama cuando se guardan cambios en los datos del comerciante.
- **createComercianteObject:**
  - Descripción: Crea un objeto con los datos del comerciante para enviar en la solicitud PUT.
  - Uso: Forma el objeto con los datos personales del comerciante.
- **createComercioObject:**
  - Descripción: Crea un objeto con los datos del comercio para enviar en la solicitud PUT.
  - Uso: Forma el objeto con los detalles comerciales del comercio.
- **makePutApiCall:**
  - Descripción: Realiza una solicitud HTTP PUT genérica.
  - Uso: Usado internamente por updateComerciante para hacer llamadas PUT a la API.
- **handleUpdateSuccess:**
  - Descripción: Maneja la respuesta exitosa de la solicitud PUT, mostrando un mensaje de éxito.
  - Uso: Se ejecuta después de recibir una respuesta exitosa de la API al actualizar.
- **handleUpdateError:**

- Descripción: Maneja los errores de la solicitud PUT, mostrando un mensaje de error.
- Uso: Se ejecuta en caso de un error en la solicitud PUT.

- **setPhones:**
  - Descripción: Gestiona los cambios en los números de teléfono y los almacena en el estado.
  - Uso: Se llama al cambiar los números de teléfono.
- **handleTabClick:**
  - Descripción: Maneja el clic en las pestañas para cambiar la pestaña activa.
  - Uso: Cambia la pestaña activa en respuesta al clic del usuario.
- **toggleMenuTab1 y toggleMenuTab2:**
  - Descripción: Controla la visibilidad de los menús contextuales en las pestañas.
  - Uso: Se utilizan para abrir o cerrar los menús contextuales.
- **handleOptionClickTab1 y handleOptionClickTab2:**
  - Descripción: Maneja el clic en las opciones del menú contextual para cada pestaña.
  - Uso: Cambia la visibilidad de botones y habilita la edición de campos.
- **handleSaveClickTab1:**
  - Descripción: Maneja el clic en el botón de guardar en la primera pestaña.
  - Uso: Realiza validaciones y actualiza los datos del comerciante en la API.
- **checkIfMetrajeIsValid:**
  - Descripción: Realiza una validación del formato del campo de metraje.
  - Uso: Se llama antes de actualizar para garantizar un formato válido.
- **handleCancelClickTab1 y handleCancelClickTab2:**
  - Descripción: Maneja el clic en el botón de cancelar cambios en las pestañas.
  - Uso: Cancela los cambios y deshabilita la edición de campos.
- **cancelarCambios:**

- Descripción: Revierte los cambios si se han realizado cambios no guardados.
- Uso: Se llama al hacer clic en el botón de cancelar cambios.

- **handleDeleteClick:**
  - Descripción: Maneja el clic en el botón de eliminar comerciante, mostrando un mensaje de confirmación.
  - Uso: Se utiliza para eliminar un comerciante tras la confirmación del usuario.
- **handleClasificacionChange,handleHorarioChange, handleTipoChange:**
  - Descripción: Maneja cambios en las opciones seleccionadas en los menús desplegables.
  - Uso: Actualiza el estado en respuesta a cambios de selección.
- **openModal y closeModal:**
  - Descripción: Controla la visibilidad del modal en la vista.
  - Uso: Abre y cierra el modal según la interacción del usuario.
- **handleInputChange, handlePhoneOneChange, handlePhoneTwoChange, handleTerceraEdadChange:**
  - Descripción: Maneja cambios en los campos de entrada y estados relacionados.
  - Uso: Se utilizan para actualizar el estado en respuesta a cambios en la entrada del usuario.
- **navigateToOrdenDePago y volver:**
  - Descripción: Navega a la página de orden de pago o regresa al panel de control.
  - Uso: Se utiliza para la navegación entre diferentes secciones de la aplicación.
- **VerificarFormatoBaja:**
  - Descripción: Realiza una verificación antes de dirigirse a la página de formato de baja del comercio.
  - Uso: Se utiliza antes de navegar a la página de formato de baja para asegurarse de que la operación sea válida.

Renderizado JSX:

Sección de Información del Comerciante:

- Formulario (form)
    - Sección 1 (col-span-1 px-4 mb-6 md:mb-0):
        - Título "Información"
        - Campos de entrada y selects para la información personal del comerciante (nombre, género, fecha de nacimiento, etc.).

Sección de Domicilio del Comerciante:

- Formulario (form)
    - Sección 2 (col-span-1 px-4):
        - Título "Domicilio"
        - Campos de entrada para la dirección del comerciante (calle, número exterior, número interior, colonia, código postal, municipio, observaciones).

Sección de Información y Ubicación del Comercio:

- Formulario (form)
    - Sección 1 (col-span-1 px-4 mb-6 md:mb-0):
        - Título "Información"
        - Campos de entrada y selects para información relacionada con el comercio (clasificación, fecha de inicio, fecha de término, tipo de giro, metros, horario, tipo, días trabajados, observaciones del comercio).
    - Sección 2 (col-span-1 px-4):
        - Título "Ubicación del comercio"
        - Campos de entrada para la dirección del comercio (calle, calles colindantes, localidad, zona).

Botones Guardar y Cancelar:

- Condicionalmente renderizados según la variable showButtonsTab1 o showButtonsTab2.

- Botón "Guardar" y "Cancelar" con funciones asociadas (handleSaveClickTab1, handleCancelClickTab1, handleSaveClickTab2, handleCancelClickTab2).

Pie de Página:

- Condicionalmente renderizado según la ausencia de botones en ambas secciones.
- Botón "Volver" para regresar.

Esta estructura representa un componente reactivo que se ajusta dinámicamente a las variables showButtonsTab1 y showButtonsTab2. Cada sección contiene elementos de formulario y títulos descriptivos para facilitar la comprensión y edición de la información del comerciante y su comercio.

6.3.3 NewComerciante:

Descripción:

La vista NewComercianteView es un componente de React que forma parte de la interfaz de usuario para registrar nuevos comerciantes. Este componente utiliza el paquete react-hook-form para gestionar formularios de manera efectiva. Aquí hay una descripción detallada del código:

Importaciones:

- React, { useEffect }: Importa React y el hook useEffect para gestionar efectos secundarios en el componente.
- useForm: Importa el hook useForm de react-hook-form para manejar los formularios de manera eficiente.
- Header: Componente de encabezado utilizado en la aplicación.
- useLocation, useNavigate: Importa los hooks de react-router-dom para gestionar la ubicación y navegación en la aplicación.
- AuthMiddleware: Middleware encargado de la autenticación del usuario.
- Swal: Importa el componente Swal de SweetAlert2 para mostrar alertas visuales.

Estado y Variables:

- location: Obtiene la ubicación actual de la aplicación.
- data: Extrae los datos de la ubicación, como el usuario y el comerciante.
- user: Almacena la información del usuario, si está presente.
- comerciante: Almacena la información del comerciante, si está presente.
- navigate: Función para la navegación en la aplicación.
- register, handleSubmit, watch: Métodos proporcionados por useForm para manejar la lógica del formulario.
- errors: Estado que contiene los errores del formulario.

Funciones:

- onSubmit: Función que se ejecuta al enviar el formulario. Realiza validaciones en los campos de teléfono y código postal. Muestra alertas si hay errores. Si no hay errores, imprime los datos del formulario y navega a la siguiente página.

Renderizado JSX:

- Contiene un encabezado (Header).
- Incluye un formulario con secciones para datos personales y de dirección del comerciante.
- Utiliza clases de estilo de Tailwind CSS para el diseño y estilos de los elementos.
- Incluye botones para volver a la página del panel de control del ejecutivo o para avanzar al siguiente paso.

Resumen:

NewComercianteView es un componente funcional de React que proporciona una interfaz de usuario para registrar nuevos comerciantes. Utiliza react-hook-form para gestionar formularios y presenta una interfaz interactiva y visualmente atractiva. Además, incluye validaciones y alertas para mejorar la experiencia del usuario durante el proceso de registro.

6.3.4 NewCommerce:

Descripción:

La vista NewComercioView es otro componente de React que forma parte del proceso de registro de nuevos comercios. Aquí hay una descripción detallada del código:

Importaciones:

- React, { useState }: Importa React y el hook useState para gestionar el estado local del componente.
- useForm: Importa el hook useForm de react-hook-form para manejar los formularios de manera eficiente.
- Header: Componente de encabezado utilizado en la aplicación.
- useLocation, useNavigate: Importa los hooks de react-router-dom para gestionar la ubicación y navegación en la aplicación.
- showInfoAlert: Importa la función showInfoAlert desde el componente SwAlerts.
- axios: Importa Axios para realizar solicitudes HTTP y gestionar la comunicación con el backend.
- Swal: Importa el componente Swal de SweetAlert2 para mostrar alertas visuales.

Estado y Variables:

- navigate: Función para la navegación en la aplicación.
- location: Obtiene la ubicación actual de la aplicación.
- comercianteData: Almacena la información del comerciante desde la ubicación.
- register, handleSubmit, watch: Métodos proporcionados por useForm para manejar la lógica del formulario.
- errors: Estado que contiene los errores del formulario.

Funciones:

- onSubmit: Función que se ejecuta al enviar el formulario. Realiza validaciones en el campo de metraje y realiza una solicitud POST al backend

para crear un nuevo comercio. Muestra alertas visuales según el resultado de la operación.

Renderizado JSX:

- Contiene un encabezado (Header).
- Incluye un formulario con secciones para la información y la dirección del comercio.
- Utiliza clases de estilo de Tailwind CSS para el diseño y estilos de los elementos.
- Incluye botones para volver a la página del panel de control del ejecutivo o para avanzar al siguiente paso.

Resumen:

NewComercioView es un componente funcional de React que proporciona una interfaz de usuario para registrar nuevos comercios. Utiliza react-hook-form para gestionar formularios y presenta una interfaz interactiva y visualmente atractiva. Además, incluye validaciones y alertas para mejorar la experiencia del usuario durante el proceso de registro. La comunicación con el backend se realiza mediante solicitudes Axios, y se utilizan alertas visuales de SweetAlert2 para informar sobre el resultado de las operaciones.

6.3.5 OrdenPago:

Descripción:

La vista OrdenPago es un componente de React que gestiona el proceso de creación de una orden de pago. Aquí se presenta una descripción detallada del código:

Importaciones:

- React, { useEffect, useState }: Importa React y los hooks useEffect y useState para gestionar el ciclo de vida del componente y el estado local.
- Header: Componente de encabezado utilizado en la aplicación.
- Select: Componente de selección de React utilizado para elegir un concepto de pago.

- DatePickerInput: Componente de entrada de fecha utilizado para seleccionar fechas.
- CheckboxInput: Componente de entrada de casilla de verificación utilizado para seleccionar días de la semana.
- Link, useLocation, useNavigate: Importa varios elementos de react-router-dom para gestionar la navegación y la ubicación en la aplicación.
- axios: Importa Axios para realizar solicitudes HTTP y gestionar la comunicación con el backend.
- ConceptList: Componente que muestra la lista de conceptos de pago.
- showErrorAlert, showInfoAlertOp: Funciones importadas desde el componente SwAlerts para mostrar alertas visuales.
- setDateFormat: Función importada desde el componente formatDates para dar formato a las fechas.
- DisableableButton: Componente de botón deshabilitable utilizado en el formulario.

Estado y Variables:

- Varios estados: Se utilizan múltiples estados para gestionar datos como opciones de conceptos, días seleccionados, fechas de inicio y fin, conceptos de pago, y más.
- Data, merchant, shop, phone: Variables que almacenan información del comerciante y su tienda obtenida de la ubicación.

Funciones:

- handleChange: Función que maneja el cambio en la selección de un concepto de pago.
- agregaConceptoPago: Función que agrega un concepto de pago a la lista, realizando validaciones.
- isValidForm: Función que valida si el formulario es válido antes de agregar un concepto.
- addConceptoPago: Función que agrega un concepto de pago a la lista.

- pushConceptosPago: Función que actualiza la lista de conceptos agregados y recalcula el total.
- popConceptosPago: Función que elimina un concepto de pago de la lista y ajusta el total.
- createPaymentOrder: Función que realiza una solicitud POST al backend para crear una orden de pago y navegar a la página de generación de PDF.

Renderizado JSX:

- Se muestra un encabezado (Header).
- Se presenta un formulario dividido en dos secciones, mostrando información sobre el contribuyente y el comercio.
- Se utiliza el componente Select para elegir un concepto de pago.
- Se incluyen elementos de entrada para seleccionar fechas y días de la semana.
- Se muestra una lista de conceptos de pago agregados.
- Se presenta una tabla que resume los detalles de los conceptos de pago.
- Se muestra un pie de página con botones para volver y generar la orden de pago.

Resumen:

OrdenPago es un componente funcional de React que facilita la creación de una orden de pago. Utiliza varios componentes, estados y funciones para gestionar la entrada de datos del usuario, realizar validaciones y comunicarse con el backend. La interfaz de usuario es intuitiva y visualmente atractiva, mejorando la experiencia del usuario durante el proceso de creación de la orden de pago.

6.4 PDFViews:

6.4.1 BajaCommerce:

Descripción:

La vista BajaCommerce es un componente de React diseñado para mostrar y permitir la impresión de un formato de baja. Aquí está una descripción detallada del código:

Importaciones:

- React, { useEffect, useRef, useState }: Importa React y los hooks useEffect, useRef, y useState para gestionar el ciclo de vida del componente y el estado local.
- cintillo: Importa una imagen utilizada en la vista.
- useReactToPrint: Importa el hook useReactToPrint de la librería 'react-to-print' para facilitar la impresión.
- useParams, useNavigate: Importa los hooks useParams y useNavigate de 'react-router-dom' para obtener parámetros de la URL y gestionar la navegación.
- axios: Importa Axios para realizar solicitudes HTTP y gestionar la comunicación con el backend.

Estado y Variables:

- {folio}: Obtiene el parámetro 'folio' de la URL utilizando el hook useParams.
- [data, setData]: Utiliza el estado data para almacenar la información obtenida del backend mediante una solicitud HTTP.

Funciones:

- useEffect: Realiza una solicitud al backend para obtener la información necesaria cuando el componente se monta. Utiliza el folio de los parámetros de la URL para realizar la solicitud.
- generatePDF: Utiliza el hook useReactToPrint para generar un documento PDF del contenido del componente TramiteImprimible. Este hook recibe una función que devuelve el contenido a imprimir.

Renderizado JSX:

- Muestra un encabezado con el título "Formato de baja".
- Incluye botones para imprimir y regresar al inicio.
- Contiene un contenedor que envuelve el contenido que será impreso. Este contenedor hace referencia a componentRef.
- Renderiza el componente TramiteImprimible que contiene el contenido a imprimir.

- El contenido a imprimir incluye una imagen, información sobre el jefe de mercados, la razón de la solicitud de baja, y un espacio para la firma y el nombre del solicitante.
- Se utiliza información obtenida del backend (nombre, giro, domicilio, colonia) para llenar dinámicamente partes del documento.

Resumen:

BajaCommerce es un componente de React que se encarga de mostrar y permitir la impresión de un formato de baja. Utiliza datos dinámicos del backend y facilita la generación de un documento PDF a través del uso del hook useReactToPrint. Este componente proporciona una interfaz clara y permite a los usuarios imprimir el formato de baja fácilmente.

6.4.2 CedulaCommerce:

Descripción:

La vista Cedulacommerce es un componente de React diseñado para mostrar y permitir la impresión de una cédula de comercio. Aquí está una descripción detallada del código:

Importaciones:

- React, { useEffect, useRef, useState }: Importa React y los hooks useEffect, useRef, y useState para gestionar el ciclo de vida del componente y el estado local.
- cintillo: Importa una imagen utilizada en la vista.
- logoEmpresa: Importa el logo de la empresa.
- layuoutp: Importa una imagen de layout para la empresa.
- useReactToPrint: Importa el hook useReactToPrint de la librería 'react-to-print' para facilitar la impresión.
- useParams, useNavigate: Importa los hooks useParams y useNavigate de 'react-router-dom' para obtener parámetros de la URL y gestionar la navegación.
- axios: Importa Axios para realizar solicitudes HTTP y gestionar la comunicación con el backend.

- setDateFormat: Importa una función setDateFormat desde un archivo que probablemente formatea fechas.

Estado y Variables:

- {folio}: Obtiene el parámetro 'folio' de la URL utilizando el hook useParams.
- [data, setData]: Utiliza el estado data para almacenar la información obtenida del backend mediante una solicitud HTTP.
- [vigencia, setVigencia]: Estado para almacenar la fecha de vigencia después de formatearla.
- [expedicion, setExpedicion]: Estado para almacenar la fecha de expedición después de formatearla.

Funciones:

- useEffect: Realiza una solicitud al backend para obtener la información necesaria cuando el componente se monta. Utiliza el folio de los parámetros de la URL para realizar la solicitud. Formatea las fechas y almacena la información en el estado.
- generatePDF: Utiliza el hook useReactToPrint para generar un documento PDF del contenido del componente TramiteImprimible. Este hook recibe una función que devuelve el contenido a imprimir.

Renderizado JSX:

- Muestra un encabezado con el título "Cédula de comercio".
- Incluye botones para imprimir y regresar al inicio.
- Contiene un contenedor que envuelve el contenido que será impreso. Este contenedor hace referencia a componentRef.
- Renderiza el componente TramiteImprimible que contiene el contenido a imprimir.
- El contenido a imprimir incluye imágenes, información sobre el comercio (nombre, clasificación, vigencia, folio, giro, ubicación, zona, metros, horario, observaciones, fecha de expedición), y un mensaje sobre las restricciones de la cédula.

- Utiliza información obtenida del backend para llenar dinámicamente partes del documento.
- Incluye un mensaje sobre las restricciones de la cédula.
- Muestra los logos de la empresa y el layout.

Resumen:

Cedulacommerce es un componente de React que se enfoca en la presentación y generación de una cédula de comercio para su impresión. Utiliza datos dinámicos del backend y facilita la generación de un documento PDF a través del uso del hook useReactToPrint. Este componente proporciona una interfaz clara y permite a los usuarios imprimir la cédula de comercio fácilmente.

6.4.3 OrdenPago:

Descripción:

OrdenPago es un componente de React diseñado para mostrar y permitir la impresión de una orden de pago relacionada con el comercio en la vía pública. A continuación, se detallan los aspectos clave del código:

Importaciones:

El componente importa las bibliotecas y recursos necesarios, incluyendo React, varios hooks de React para el manejo del estado y efectos, imágenes y bibliotecas externas como Axios para manejar solicitudes HTTP.

Estado y Variables:

- location, referencia, idComercio: Extraen información de la ubicación actual y parámetros relacionados.
- componentRef: Una referencia a un componente de React utilizado para la impresión.
- data, direccion, fecha_actual, diaDeLaSemana, unidades: Estados para almacenar información obtenida del backend, detalles de dirección, fechas, días de la semana y unidades.

Funciones:

- useEffect: Utiliza este hook para realizar una solicitud al backend al montar el componente. Los datos obtenidos se formatean y se almacenan en el estado local.
- generatePDF: Utiliza useReactToPrint para generar un documento PDF del contenido del componente TramiteImprimible.

Renderizado JSX:

- Encabezado e Interfaz de Usuario: Muestra un encabezado con el título "Orden de pago vía pública". Incluye botones para imprimir y regresar al inicio.
- Contenido a Imprimir: Utiliza un contenedor que envuelve el contenido que se imprimirá. Renderiza el componente TramiteImprimible que contiene detalles específicos de la orden de pago.
- Tabla y Detalles: La tabla presenta detalles específicos de costos y cantidades relacionadas con el comercio.
- Firma y Logos: Incluye información sobre la firma autorizada y logos de la empresa.
- Mensajes Adicionales y Resumen: Se incluyen mensajes adicionales sobre el proceso de autorización y se proporcionan detalles clave del comercio.
- Formato Duplicado: El contenido se repite dos veces, sugiriendo un formato para imprimir en una página y doblarla.

Resumen:

OrdenPago proporciona una interfaz clara y estructurada para visualizar y generar documentos PDF de órdenes de pago relacionadas con el comercio en la vía pública. Utiliza datos dinámicos del backend y facilita la generación de documentos PDF para su posterior impresión.

6.4.4 TerceraEdad:

Descripción:

El componente TerceraEdad es una vista de React diseñada para mostrar y permitir la impresión de un formato relacionado con beneficios fiscales para personas de

tercera edad o con discapacidad. Aquí hay un resumen de los aspectos clave del código:

Importaciones:

Se importan las bibliotecas y recursos necesarios, como React, varios hooks de React, imágenes y bibliotecas externas como Axios y react-to-print para manejar solicitudes HTTP e impresión respectivamente.

Estado y Variables:

- **componentRef, id_comerciante, dataMerchant, fecha**: Variables de estado y referencia utilizadas para almacenar datos del comerciante, su identificación, la fecha de término y una referencia al componente de React utilizado para la impresión.

Efecto y Solicitudes HTTP:

- Se utiliza el useEffect para realizar una solicitud al backend al montar el componente. Los datos obtenidos se almacenan en el estado local.

Funciones:

- **generatePDF**: Utiliza useReactToPrint para generar un documento PDF del contenido del componente TramiteImprimible.

Renderizado JSX:

- Encabezado e Interfaz de Usuario: Muestra un título y botones para imprimir y regresar al inicio.
- Contenido a Imprimir: Utiliza un contenedor que envuelve el contenido que se imprimirá. Renderiza el componente TramiteImprimible que contiene detalles específicos del formato.
- Datos del Comerciante: Presenta información detallada sobre el comerciante, incluyendo nombre, ubicación, giros, horario, metros y vigencia.
- Firmas y Mensajes Adicionales: Incluye firmas y mensajes adicionales relacionados con la autorización del beneficio fiscal y la revocación del permiso.

Resumen:

TerceraEdad proporciona una interfaz clara y estructurada para visualizar y generar documentos PDF de formatos relacionados con beneficios fiscales para personas de tercera edad o con discapacidad. Utiliza datos dinámicos del backend y facilita la generación de documentos PDF para su posterior impresión.

7. Recursos Adicionales

A continuación, se presentan algunos recursos adicionales que pueden resultar útiles para los desarrolladores que trabajan en el proyecto del cliente:

7.1 Documentación Oficial

- React: Documentación oficial de React
- Tailwind CSS: Documentación oficial de Tailwind CSS
- Node.js: Documentación oficial de Node.js
- Next.js: Documentación oficial de Next.js

Estos recursos proporcionan información detallada, donde los desarrolladores pueden buscar ayuda adicional. Explorarlos puede ser beneficioso para comprender más a fondo las tecnologías utilizadas en el proyecto del cliente y abordar posibles desafíos durante el desarrollo.

8. Conclusión

En la culminación de este manual técnico del cliente, hemos explorado exhaustivamente la estructura, herramientas y detalles de implementación que componen la aplicación. Desde la organización de carpetas hasta los componentes reutilizables y las vistas específicas, este manual proporciona una guía detallada para comprender y trabajar con el cliente.

La combinación de tecnologías como React, Tailwind CSS, y las prácticas de desarrollo de Node.js ha permitido la creación de una interfaz de usuario dinámica y eficiente. La modularidad y reutilización de código se han enfatizado a través de

componentes y ganchos personalizados, facilitando el mantenimiento y la expansión futura de la aplicación.

Este manual no solo sirve como referencia técnica, sino también como un recurso valioso para aquellos que participarán en el desarrollo, mantenimiento y comprensión del cliente. Con esta sólida base técnica, estamos preparados para enfrentar los desafíos que se presenten y seguir mejorando esta aplicación en el futuro.