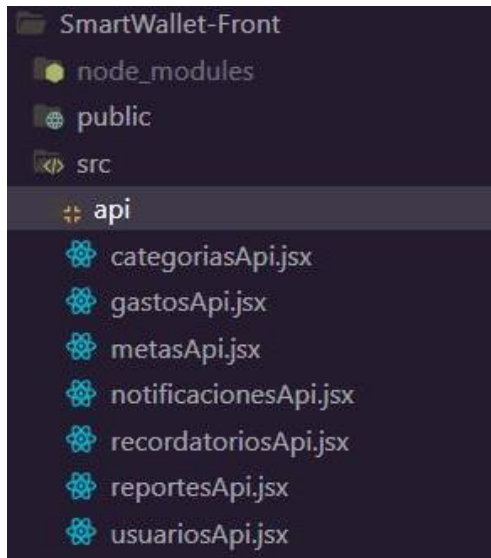


Documentaci6n del Frontend de Smart Wallet

1. Estructura del Proyecto

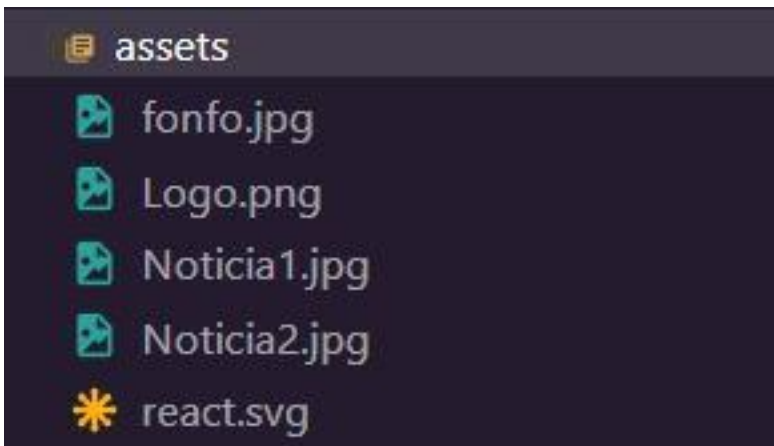
El proyecto est estructurado en varias carpetas y archivos que organizan las diferentes funcionalidades y componentes del sistema. A continuaci6n, se presenta la estructura principal:

- src/: Contiene todos los archivos fuente de la aplicaci6n.
 - api/: Archivos que manejan la comunicaci6n con la API.
 - categoriasApi.jsx: Maneja las solicitudes relacionadas con las categoras.
 - gastosApi.jsx: Maneja las solicitudes relacionadas con los gastos.
 - metasApi.jsx: Maneja las solicitudes relacionadas con las metas de ahorro.
 - notificacionesApi.jsx: Maneja las solicitudes relacionadas con las notificaciones.
 - recordatoriosApi.jsx: Maneja las solicitudes relacionadas con los recordatorios.
 - reportesApi.jsx: Maneja las solicitudes relacionadas con los reportes.
 - usuariosApi.jsx: Maneja las solicitudes relacionadas con los usuarios.



assets/: Contiene imgenes y archivos estticos utilizados en la aplicaci6n.

- fonfo.jpg: Imagen de fondo.
- Logo.png: Logo de la aplicaci6n.
- Noticia1.jpg, Noticia2.jpg: imgenes de noticias.
- react.svg: icono de React.



FrontEnd

Dentro de la carpeta de Componenets nos podemos encontrar con múltiples carpetas las cuales cuentan con componen entes para crear la interfaz de usuario, dentro de estas carpetas se encuentran la carpeta de Componentes, Modales, Charts y Protected. Esta es la estructura que se muestra a continuación

```
└─ CardMeta.jsx
└─ CardReport.jsx
└─ CongratulationsModal.jsx
└─ Dialog.jsx
└─ EditableField.jsx
└─ Footer.jsx
└─ GastosList.jsx
└─ GlobalNotification.jsx
└─ Header.jsx
└─ HeaderAdmin.jsx
└─ InformationCard.jsx
└─ Input.jsx
└─ MetasList.jsx
└─ Navbar.jsx
└─ NotificacionesList.jsx
└─ RecentExpensesTable.jsx
└─ RemindersList.jsx
└─ scroll-area.jsx
└─ SelectAccountType.jsx
└─ Sidebar.jsx
└─ SidebarAdmin.jsx
└─ table.jsx
└─ UserProfileCard.jsx
└─ button.tsx
└─ HomePage.tsx
└─ input.tsx
```

La carpeta pages contiene componentes que representan diferentes páginas de la aplicación Smart Wallet. Cada uno de estos componentes se encarga de una funcionalidad específica y presenta la interfaz de usuario correspondiente.

Descripción de Componentes

1. AboutSection.jsx

- Descripción: Componente que muestra información sobre el proyecto Smart Wallet, sus objetivos y su propósito.
- Uso: Este componente se puede incluir en la página principal o en secciones informativas.

Archive: AboutSection.jsx

Este componente de React representa una sección de "Sobre Nosotros" para la página de una aplicación llamada SmartWallet. Incluye un encabezado con fondo de imagen, una descripción general de la empresa y tarjetas interactivas que detallan la misión, la historia y la estructura de liderazgo de la empresa. Cada tarjeta contiene un botón que abre un modal con más información.

Dependencias

- React: Librería para la creación de interfaces de usuario.
- PropTypes: Verifica los tipos de datos de las props pasadas a los componentes.
- react-router-dom: Proporciona la navegación entre páginas en aplicaciones de React.
- lucide-react: Biblioteca de iconos de React.
- framer-motion: Biblioteca para animaciones y transiciones en React.
- Componentes personalizados:
 - Navbar: Barra de navegación.
 - InfoModal: Modal para mostrar información detallada.
 - Footer: Pie de página.

Estructura del Componente AboutSection

1. Navbar: Componente de la barra de navegación que se muestra en la parte superior.
2. Sección de Encabezado:
 - Un contenedor animado con una imagen de fondo y un título.
3. Contenido Principal:
 - Un contenedor con tarjetas (AboutCard), cada una representando una parte clave de la empresa.
4. Footer: Componente de pie de página al final de la sección.

Propiedades y Funcionalidades del Componente AboutCard

- icon: Prop que recibe un icono JSX para representar visualmente el tema de la tarjeta.
- title: Prop que representa el título de la tarjeta.
- description: Prop que contiene una breve descripción de la información de la tarjeta.
- modalContent: Prop que contiene el contenido detallado que se muestra en el modal al hacer clic en el botón.

Comportamiento de la Tarjeta

- Cada tarjeta se anima al aparecer en la página mediante framer-motion.
- Tiene un botón que, al hacer clic, abre un modal (InfoModal) con información más detallada.
- La tarjeta y el modal están diseñados con estilos de Tailwind CSS.

Modal InfoModal

- Este modal se muestra al usuario al hacer clic en "Mas información" en una tarjeta.
- isOpen: Estado de visibilidad del modal.
- onClose: Función que cierra el modal.
- title y content: Props que definen el título y contenido del modal, respectivamente

```
import React from 'react';
import { motion } from 'framer-motion';
import { useNavigate } from 'react-router-dom';
import { ArrowRight, Info, Mail, Phone } from 'lucide-react';
import { AboutCard } from './AboutCard';
import { InfoModal } from './InfoModal';

const AboutSection = () => {
  const navigate = useNavigate();
  const [isOpen, setIsOpen] = React.useState(false);
  const [selectedCard, setSelectedCard] = React.useState(null);

  const handleCardClick = (card) => {
    setSelectedCard(card);
    setIsOpen(true);
  };

  const closeModal = () => {
    setIsOpen(false);
  };

  return (
    <div>
      <div>
        <h2>Sobre SmartWallet</h2>
        <p>SmartWallet es una aplicación revolucionaria que simplifica la gestión de tus finanzas personales. Con una interfaz intuitiva y herramientas avanzadas, puedes controlar tus gastos, ahorrar dinero y tomar decisiones financieras más informadas. Únete a miles de usuarios que ya están disfrutando de la tranquilidad que SmartWallet ofrece.</p>
      </div>
      <div>
        <div>
          <div>
            <img alt="Icon representing Mission" data-bbox="500 930 515 945"/>
            <h3>Nuestra Misión</h3>
            <p>Proporcionar una experiencia financiera simplificada y segura para todos, ayudando a mejorar su calidad de vida.</p>
          </div>
          <div>
            <img alt="Icon representing History" data-bbox="615 930 630 945"/>
            <h3>Nuestra Historia</h3>
            <p>SmartWallet nació de la necesidad de una herramienta que ayudara a las personas a gestionar sus finanzas de manera más eficiente y segura.</p>
          </div>
          <div>
            <img alt="Icon representing Structure" data-bbox="725 930 740 945"/>
            <h3>Estructura de Liderazgo</h3>
            <p>SmartWallet cuenta con un equipo de expertos en finanzas y tecnología, liderados por un equipo de profesionales de la industria.</p>
          </div>
        </div>
      </div>
    </div>
  );
};

export default AboutSection;
```



AdminOverviewPage.jsx

- Descripción: Página de administración que proporciona un resumen de las estadísticas y datos relevantes para los administradores.
- Uso: Permite a los administradores obtener una visión general de la actividad de la aplicación.

Archivo: AdminOverviewPage.jsx

Este componente de React muestra una página de vista general para los administradores, proporcionando métricas clave sobre los usuarios registrados y una gráfica que visualiza el registro de usuarios por mes. Incluye un diseño dividido con una barra lateral, encabezado y un área principal que muestra tarjetas de métricas y gráficos.

Dependencias

- React: Usa hooks (useState, useEffect) para gestionar el estado y cargar datos asíncronamente.
- obtenerInfoUsuarios: Función importada desde usuariosApi que obtiene datos de usuarios desde una API.
- SidebarAdmin: Componente de barra lateral para navegación.
- HeaderAdmin: Componente de encabezado para la vista del administrador.
- recharts: Biblioteca de gráficos utilizada para crear un gráfico de líneas.
- AdminCard: Componente de tarjeta que muestra métricas y gráficos.

Estado del Componente

- usuariosRegistrados: Almacena el total de usuarios registrados.
- usuariosRegistradosMes: Almacena el número de usuarios registrados en el mes actual.
- graficaData: Datos para el gráfico de líneas de usuarios registrados mensualmente.
- loading: Estado booleano que indica si los datos están en proceso de carga.

useEffect

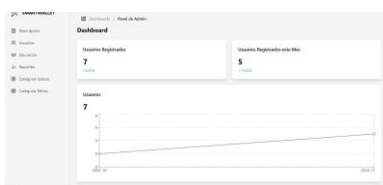
Utilizado para cargar los datos de usuarios al montar el componente. La función asíncrona fetchData llama a obtenerInfoUsuarios y actualiza los estados usuariosRegistrados, usuariosRegistradosMes, y graficaData con la información obtenida de la API. Maneja errores de carga y actualiza el estado de loading una vez que los datos están listos o ha ocurrido un error.

Renderizado Principal

1. SidebarAdmin: Barra lateral fija para navegación del administrador.
2. HeaderAdmin: Encabezado que proporciona contexto y herramientas adicionales para la administración.
3. Título: Encabezado principal de la página ("Dashboard").
4. Tarjetas de Métricas:
 - Usuarios Registrados: Tarjeta que muestra el total de usuarios registrados.
 - Usuarios Registrados este Mes: Tarjeta que muestra el número de usuarios registrados en el mes actual.
5. Gráfico de Usuarios:
 - Utiliza ResponsiveContainer de recharts para un gráfico adaptable que muestra la evolución de usuarios registrados por mes.
 - El gráfico de líneas (LineChart) incluye XAxis, YAxis, Tooltip, y una cuadrícula (CartesianGrid) para mejorar la visualización de los datos.
 - La línea muestra el número de usuarios registrados cada mes (dataKey="value").

Componentes y Propiedades del Gráfico

- ResponsiveContainer: Asegura que el gráfico se ajuste al contenedor.
- LineChart: Gráfico de líneas que utiliza graficaData como fuente de datos.
- XAxis y YAxis: Ejes que indican los datos de meses y el número de usuarios, respectivamente.
- Tooltip: Muestra detalles al pasar el cursor sobre un punto en el gráfico.
- Line: Representa la serie de datos en el gráfico, con estilo personalizado y puntos interactivos.



```
import { useState, useEffect } from 'react';
import { LineChart, XAxis, YAxis, Tooltip, CartesianGrid } from 'recharts';
import { obtenerInfoUsuarios } from 'api';
import { AdminCard } from 'components';

export default function AdminOverviewPage() {
  const [usuariosRegistrados, setUsuariosRegistrados] = useState(0);
  const [usuariosRegistradosMes, setUsuariosRegistradosMes] = useState(0);
  const [graficaData, setGraficaData] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const { totalUsuarios, usuariosMes, graficaUsuarios } =
          await obtenerInfoUsuarios();

        setUsuariosRegistrados(totalUsuarios);
        setUsuariosRegistradosMes(usuariosMes);
      } catch (error) {
        console.error('Error al cargar los datos');
      }
    };

    fetchData();
  }, []);

  return (
    <div>
      <HeaderAdmin />
      <SidebarAdmin />
      <div>
        <h2>Dashboard</h2>
        <div>
          <AdminCard
            title="Usuarios Registrados"
            value={usuariosRegistrados}
            icon="users"
          />
          <AdminCard
            title="Usuarios Registrados este Mes"
            value={usuariosRegistradosMes}
            icon="calendar"
          />
        </div>
        <div>
          <h3>Usuarios Registrados por Mes</h3>
          <LineChart
            data={graficaData}
            margin={{ top: 10, right: 10, bottom: 10, left: 10 }}
          />
        </div>
      </div>
    </div>
  );
}
```

Archive: AdminReportsPage.jsx

Este componente de React representa una página de administración para gestionar reportes. Permite a los administradores ver, buscar, paginar y eliminar reportes de la base de datos a través de una interfaz interactiva.

Dependencias

- React: Utiliza los hooks `useState` y `useEffect` para gestionar el estado y manejar la carga de datos de manera asíncrona.
- `obtenerReportes`, `eliminarReporte`: Funciones importadas de `reportesApi` para obtener y eliminar reportes del backend.
- `Input`: Componente de entrada personalizado.
- `ScrollArea`: Componente de área de desplazamiento para mostrar una lista de reportes.
- `HeaderAdmin`, `SidebarAdmin`: Componentes de UI que representan el encabezado y la barra lateral de la página de administración.
- `CardReport`: Componente para mostrar la información de cada reporte.
- `Button`: Componente de botón personalizado.

Estado del Componente

- `reports`: Arreglo que contiene todos los reportes cargados desde el backend.
- `filteredReports`: Arreglo que contiene los reportes filtrados en función de la búsqueda.
- `searchTerm`: Cadena de texto que representa el término de búsqueda ingresado por el usuario.
- `isConfirmModalOpen`: Booleano que indica si el modal de confirmación de eliminación está abierto.
- `reportToDelete`: ID del reporte que se va a eliminar.
- `currentPage`: Número de página actual para la paginación.
- `reportsPerPage`: Cantidad de reportes mostrados por página.
- `totalPages`: Número total de páginas calculadas.

`useEffect`

Llama a la función `fetchReports` cuando el componente se monta para cargar los reportes desde el backend. La función maneja la carga de datos y captura errores en caso de fallos en la solicitud.

Funcionalidades Principales

1. Buscar Reportes:
 - La función `handleSearch` filtra los reportes basándose en el término de búsqueda ingresado.
 - Reinicia la página a la primera cuando se realiza una búsqueda.
2. Abrir y Cerrar Modal de Confirmación:
 - `openConfirmModal`: Abre el modal de confirmación de eliminación y establece el ID del reporte a eliminar.
 - `closeConfirmModal`: Cierra el modal y limpia el estado `reportToDelete`.
3. Eliminar Reporte:
 - La función `handleDelete` envía una solicitud al backend para eliminar el reporte especificado.
 - Actualiza el estado `reports` y `filteredReports` para reflejar la eliminación en la UI.
4. Paginación:
 - Determina el rango de reportes que se muestran en función de la página actual (`currentPage`).
 - La función `paginate` actualiza `currentPage` al número de página seleccionado.

Renderizado Principal

- SidebarAdmin: Muestra la barra lateral de navegaci6n.
- HeaderAdmin: Muestra el encabezado de la p6gina.
- Barra de b6squeda: Permite al usuario buscar reportes par t6tulo.
- Lista de Reportes:
 - Usa ScrollArea para una lista desplazable de reportes.
 - Muestra cada reporte con el componente CardReport, que incluye el t6tulo, descripci6n, fecha de creaci6n y un bot6n para eliminar.
- Paginaci6n: Botones numerados para cambiar de p6gina.
- Modal de Confirmaci6n:
 - Aparece al intentar eliminar un reporte, mostrando opciones para confirmar o cancelar la acci6n.

interactividad

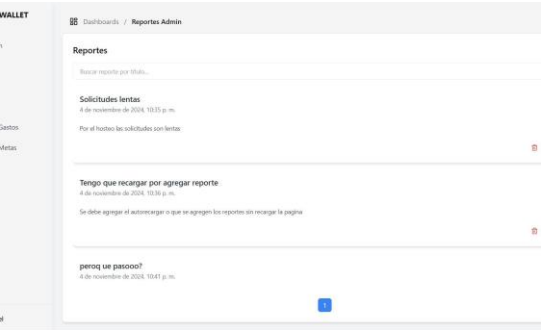
- Los botones de paginaci6n permiten la navegaci6n entre p6ginas de reportes.
- El modal confirma si el usuario quiere proceder con la eliminaci6n del reporte seleccionado.
- El componente Button maneja la acci6n de eliminar o cancelar dentro del modal.

```
import { obtenerReportes, eliminarReporte } from "../api/reportesApi"; // Asegúrate de tener `eliminarReporte`
import { Input } from "../components/ui/input";
import ScrollArea from "../components/ui/componentes/scroll-area";
import HeaderAdmin from "../components/ui/componentes/HeaderAdmin";
import SidebarAdmin from "../components/ui/componentes/SidebarAdmin";
import CardReport from "../components/ui/componentes/CardReport";
import { Button } from "../components/ui/button";

export default function AdminReportsPage() {
  const [reports, setReports] = useState([]);
  const [filteredReports, setFilteredReports] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [isConfirmModalOpen, setIsConfirmModalOpen] = useState(false);
  const [reportToDelete, setReportToDelete] = useState(null);

  // Estados de paginación
  const [currentPage, setCurrentPage] = useState(1);
  const reportsPerPage = 5;

  // Obtener los reportes desde el backend
  useEffect(() => {
    const fetchReports = async () => {
      try {
        const data = await obtenerReportes();
        setReports(data);
        setFilteredReports(data);
      } catch (error) {
        console.error("Error al cargar los reportes:", error);
      }
    }
  });
}
```



Archive: AnalysisPage.jsx

Este componente de React representa una página de análisis financiero que muestra gráficos interactivos para visualizar el estado financiero del usuario, incluyendo ingresos, gastos y progreso en las metas. La página se compone de varios gráficos que permiten al usuario ver la distribución de gastos, gastos por categoría, comparación de ingresos y gastos, y el progreso de sus metas financieras.

Dependencias

- React: Utiliza los hooks `useState`, `useEffect`, y `useContext` para manejar el estado y la carga de datos asíncrona.
- AuthContext: Contexto de autenticación para obtener el token de usuario.
- Sidebar, Header: Componentes de interfaz de usuario para la barra lateral y el encabezado de la página.
- CategoryExpensesChart, ExpensesDistributionChart, IncomeExpensesPieChart, GoalProgressChart: Componentes de gráficos para mostrar los datos financieros.
- `obtenerIngresoUsuario`, `obtenerGastosPorUsuario`, `obtenerMetasPorUsuario`: Funciones importadas de las APIs para obtener datos de ingreso, gastos y metas del usuario.

Estado del Componente

- `income`: Estado que almacena el ingreso total del usuario.
- `totalExpenses`: Estado que almacena el monto total de gastos del usuario.
- `expenseData`: Datos formateados para el gráfico de distribución de gastos.
- `expenseCategories`: Datos de las categorías de gastos con sus respectivos montos.
- `goals`: Arreglo que contiene las metas financieras del usuario, cada una con su nombre, monto objetivo y monto alcanzado.

Funciones Principales

- `parseJwt`: Decodifica el token JWT para extraer el `usuario_id` del payload.
- `fetchIncome`: Función asíncrona que obtiene el ingreso del usuario y actualiza el estado `income`.
- `fetchExpenses`: Función asíncrona que obtiene los gastos del usuario y:
 - Calcula el total de gastos y actualiza `totalExpenses`.
 - Agrupa los gastos por categoría, calcula los montos por categoría y actualiza `expenseCategories` y `expenseData`.
- `fetchGoals`: Función asíncrona que obtiene las metas del usuario y las convierte a un formato adecuado para el gráfico de progreso. Actualiza el estado `goals`.

Ciclo de Vida (`useEffect`)

El `useEffect` se ejecuta al montar el componente y cuando cambian `token` o `usuario_id`. Llama a `fetchIncome`, `fetchExpenses` y `fetchGoals` para obtener y actualizar los datos necesarios si el `usuario_id` está disponible.

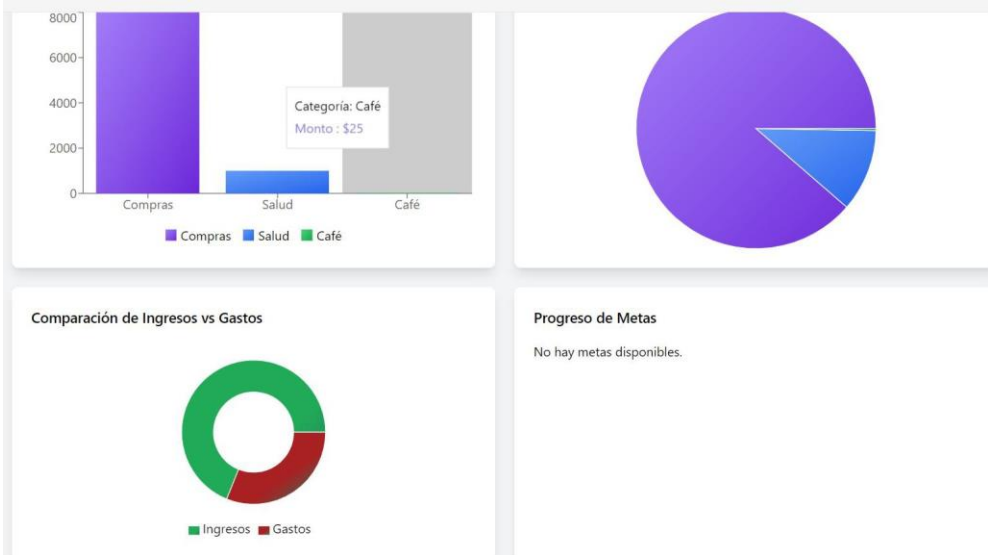
Renderizado Principal

1. Sidebar: Componente de barra lateral para la navegación.
2. Header: Componente de encabezado de la página.
3. Título: Encabezado "Análisis Financiero" en la sección principal.
4. Gráficos:
 - Gastos por Categoría: Muestra un gráfico de barras o de otro tipo con las categorías de gasto más relevantes.
 - Distribución de Gastos: Gráfico que representa la distribución de gastos en diferentes categorías.
 - Comparación de ingresos vs Gastos: Gráfico de pastel que compara el ingreso total con el gasto total.
 - Progreso de Metas: Muestra un gráfico de progreso de las metas financieras del usuario. Si no hay metas, se muestra un mensaje indicándolo.

Comportamiento de los Gráficos

- CategoryExpensesChart: Muestra las categorfas de gastos y los montos relacionados.
- ExpensesDistributionChart: Representa c6mo se distribuyen los gastos del usuario en las diferentes categorfas.
- IncomeExpensesPieChart: Compara visualmente el ingreso total del usuario con el total de gastos.
- GoalProgressChart: indica el progreso hacia las metas financieras; si no hay metas, se presenta un texto alternative.

Este componente proporciona una vista centralizada y detallada del estado financiero del usuario mediante grficos interactivos y actualizados, brindando una manera eficiente de analizar ingresos, gastos y progreso en las metas.



```
    console.error("Error al obtener gastos del usuario:", error);
  }
};

const fetchGoals = async () => {
  if (!usuario_id || !token) {
    console.warn("Token o usuarioId no disponible");
    return;
  }
  try {
    const metasUsuario = await obtenerMetasPorUsuario(usuario_id, token);
    if (metasUsuario && Array.isArray(metasUsuario)) {
      const metasConvertidas = metasUsuario.map((meta) => ({
        name: meta.nombre_meta || "Meta sin nombre",
        targetAmount: Number(meta.monto_objetivo) || 0,
        amountAchieved: Number(meta.monto_actual) || 0,
      }));
      setGoals(metasConvertidas);
    } else {
      console.warn("No se obtuvieron metas o el formato de respuesta no es correcto");
      setGoals([]);
    }
  }
}
```


Archivo: EducationAdminPage.jsx

Este componente de React representa una página de administración de educación que muestra artículos relacionados con temas financieros, cargados desde una API externa. Permite a los administradores visualizar noticias para mantenerse informados y actualizados sobre el sector financiero.

Dependencias

- React: Utiliza los hooks useState y useEffect para gestionar el estado y manejar la carga de datos de manera asíncrona.
- Button: Componente de botón personalizado.
- ScrollArea: Componente para mostrar el contenido con área de desplazamiento.
- HeaderAdmin, SidebarAdmin: Componentes de interfaz de usuario que representan el encabezado y la barra lateral de la página de administración.
- CardEducation: Componente para mostrar cada artículo de educación con detalles como título, descripción, imagen, fecha de publicación y autor.
- fetchArticles: Función importada de newsAPI para obtener los artículos desde una API externa.

Estado del Componente

- articles: Arreglo que contiene los artículos obtenidos de la API.
- loading: Booleano que indica si los datos están en proceso de carga.

useEffect

El useEffect se ejecuta al montar el componente para cargar los artículos. La función getArticles llama a fetchArticles con la categoría "finance" y actualiza el estado articles con los datos obtenidos. Maneja posibles errores en la carga y actualiza el estado loading una vez que la solicitud ha finalizado.

Funcionalidades Principales

- Carga de Artfculos:
 - Utiliza fetchArticles para obtener artículos relacionados con temas financieros.
 - Actualiza el estado articles con la respuesta de la API y controla el estado de carga (loading).

Renderizado Principal

1. SidebarAdmin: Componente de barra lateral de navegación.
2. HeaderAdmin: Componente de encabezado de la página.
3. Título: Encabezado de la sección "Educación" que introduce la página.
4. Area de Contenido Desplazable:
 - Utiliza ScrollArea para mostrar la lista de artículos con desplazamiento.
 - Si loading es true, se muestra un mensaje de carga.
 - Cuando loading es false, se mapea el estado articles y se renderiza cada artículo con CardEducation, mostrando información como título, descripción, imagen, fecha de publicación y autor.
 - Si no hay descripción o autor, se muestra un mensaje por defecto.

Comportamiento del Componente CardEducation

- title: Muestra el título del artículo.
- description: Muestra la descripción del artículo o un mensaje predeterminado si no está disponible.
- imagen: Muestra la imagen asociada al artículo.
- timeAgo: Muestra el tiempo transcurrido desde la publicación del artículo.
- autor: indica el autor del artículo o "Desconocido" si no se proporciona.
- url: Proporciona un enlace al artículo completo.



```
import React, { useState, useEffect } from 'react';
import { fetchArticles } from 'newsAPI';
import { Button } from 'react-bootstrap';
import { ScrollArea } from 'react-scroll-area';
import { HeaderAdmin } from 'HeaderAdmin';
import { SidebarAdmin } from 'SidebarAdmin';
import { CardEducation } from 'CardEducation';

const EducationAdminPage = () => {
  const [articles, setArticles] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const getArticles = async () => {
      try {
        const data = await fetchArticles("finance");
        setArticles(data);
      } catch (error) {
        console.error("Error fetching articles:", error);
      } finally {
        setLoading(false);
      }
    };
    getArticles();
  }, []);

  return (
    <div className="flex h-screen bg-gray-100">
      <SidebarAdmin />
    </div>
  );
};

export default EducationAdminPage;
```

Archivo: EducationPage.jsx

Este componente de React representa una página de educación que muestra artículos relacionados con temas financieros, obtenidos de una API externa. Ofrece a los usuarios una forma de visualizar contenido educativo sobre finanzas en un formato atractivo y fácil de navegar.

Dependencias

- React: Utiliza los hooks useState y useEffect para manejar el estado y realizar la carga de datos de forma asíncrona.
- ScrollArea: Componente que proporciona un área de desplazamiento para mostrar listas largas de contenido.
- Header, Sidebar: Componentes de la interfaz de usuario que representan el encabezado y la barra lateral de la página.
- CardEducation: Componente que muestra la información de cada artículo de educación, incluyendo título, descripción, imagen, fecha de publicación, autor, y un enlace al artículo completo.
- fetchArticles: Función importada de newsAPI para obtener artículos de noticias desde una API externa.

Estado del Componente

- articles: Arreglo que contiene los artículos obtenidos de la API.
- loading: Booleano que indica si los datos están en proceso de carga.

useEffect

El useEffect se ejecuta al montar el componente para cargar los artículos. La función getArticles llama a fetchArticles con la categoría "finance" y actualiza el estado articles con la respuesta de la API. Maneja errores durante la carga y actualiza el estado loading una vez que la solicitud se completa, ya sea con éxito o con error.

Funcionalidades Principales

- Carga de Artículos:
 - Utiliza fetchArticles para obtener artículos relacionados con finanzas.
 - Actualiza el estado articles con la información obtenida de la API.
 - Controla el estado de carga con el booleano loading para mostrar un mensaje de carga mientras los datos se obtienen.

Renderizado Principal

1. Sidebar: Componente de barra lateral para la navegación.
2. Header: Componente de encabezado de la página.
3. Título: Encabezado de la sección "Educación" que introduce la página.
4. Area de Contenido Desplazable:
 - Utiliza ScrollArea para mostrar los artículos en un área con desplazamiento.
 - Si loading es true, muestra un mensaje de carga.
 - Cuando loading es false, mapea el estado articles para renderizar cada artículo con CardEducation, mostrando información relevante como título, descripción, imagen, autor, y la fecha de publicación.
 - Si no se proporciona una descripción o autor, se muestra un texto predeterminado.

Comportamiento del Componente CardEducation

- title: Muestra el título del artículo.
- description: Muestra la descripción del artículo o un mensaje por defecto si no está disponible.
- image: Muestra la imagen asociada al artículo.
- timeAgo: Muestra la fecha de publicación del artículo.
- author: Indica el autor del artículo o "Desconocido" si no se proporciona.
- url: Proporciona un enlace al artículo completo.



```
<div className="mt-2">
  </div>
</div>

<ScrollArea className="h-[calc(100vh-120px)]">
  <div className="space-y-6">
    {loading ? (
      <p className="text-center text-gray-600">Cargando noticias...</p>
    ) : (
      articles.map((article, index) => (
        <CardEducation
          key={index}
          title={article.title}
          description={article.description || "Descripción no disponible"}
          image={article.urlToImage}
          timeAgo={article.publishedAt}
          author={article.author || "Desconocido"}
          likes={0}
          url={article.url} // Para la URL de la noticia
        </CardEducation>
      )
    )
  </div>
</ScrollArea>
```

Archivo: ExpenseCategories.jsx

Este componente de React permite la gesti6n de categorfas de gasto, donde los administradores pueden visualizar, agregar, editar y eliminar categorfas. Proporciona una interfaz interactiva con una tabla y modales para facilitar la gesti6n de datos.

Dependencias

- React: Utiliza los hooks useState y useEffect para manejar el estado y realizar operaciones as6ncronas.
- lucide-react: Biblioteca de iconos que se utiliza para mostrar iconos de acciones (e.g., agregar, editar, eliminar).
- HeaderAdmin, SidebarAdmin: Componentes de la interfaz de usuario que representan el encabezado y la barra lateral de la p6gina de administraci6n.
- Button: Componente de bot6n personalizado.
- Table, TableBody, TableCell, TableHead, TableHeader, TableRow: Componentes para crear tablas personalizadas.
- obtenerCategoriasGasto, crearCategoriaGasto, actualizarCategoriaGasto, eliminarCategoriaGasto: Funciones importadas de categoriasApi para realizar operaciones CRUD en categorfas de gasto.
- getIconForCategory: Funci6n de utilidad para obtener un icono representativo de una categorfa.
- AddCategoryModal: Componente de modal para agregar una nueva categorfa de gasto.

Estado del Componente






















- categories: Arreglo que contiene las categorfas de gasto obtenidas de la API.
- isAddModalOpen: Booleano que indica si el modal para agregar una nueva categorfa est6 abierto.
- editingCategory: Objeto que representa la categorfa que est6 siendo editada actualmente.
- confirmDelete: ID de la categorfa que est6 a punto de ser eliminada para confirmar la acci6n.

useEffect

El useEffect se ejecuta al montar el componente para cargar las categorfas de gasto llamando a la funci6n fetchCategories. La funci6n captura posibles errores en la carga de datos y actualiza el estado categories con la informaci6n obtenida de la API.

Funcionalidades Principales

- fetchCategories: Carga las categorfas de gasto desde la API y actualiza el estado categories.
- handleAddCategory: Agrega una nueva categorfa llamando a la API y actualiza la lista de categorfas al finalizar.
- startEditing: Establece la categorfa que est6 siendo editada y permite la edici6n en l6nea.
- saveCategory: Guarda los cambios en una categorfa editada enviando la actualizaci6n a la API y recarga las categorfas.
- deleteCategory: Elimina una categorfa espec6fica de la lista llamando a la API y actualiza el estado para reflejar la eliminaci6n.
- setConfirmDelete: Muestra un conjunto de botones de confirmaci6n y cancelaci6n antes de realizar la eliminaci6n.

ICONO	NOMBRE	ACCIONES
	Hogar	 
	Salud	 
	Finanzas	 
	Viaje	 
	Veh6culo	 
	Educaci6n	 
	Compras	 

Renderizado Principal

1. SidebarAdmin: Componente de barra lateral para la navegación.
2. HeaderAdmin: Componente de encabezado de la página.
3. Botón Agregar Categoría:
 - Permite abrir un modal para agregar una nueva categoría.
 - Muestra un icono de Plus y un texto de acción.
4. Tabla de Categorías:
 - Muestra una lista de categorías con columnas para icono, nombre y acciones.
 - Las acciones incluyen botones para editar (Pencil) y eliminar (Trash2).
5. Edición en Linea:
 - Permite editar el nombre de la categoría directamente en la tabla.
 - Los cambios se confirman con la tecla Enter o se cancelan haciendo clic en un botón (X).
6. Confirmación de Eliminación:
 - Muestra botones de confirmación (Check) y cancelación (X) cuando se solicita la eliminación de una categoría.
7. Modal de Agregar Categoría:
 - Se abre al hacer clic en el botón de agregar y permite al usuario ingresar un nuevo nombre de categoría.

Comportamiento del Componente AddCategoryModal

- isOpen: Controla la visibilidad del modal.
- onClose: Función que se ejecuta al cerrar el modal.
- onSave: Función que se ejecuta al guardar la nueva cate

```

const [categories, setCategories] = useState([]);
const [isAddModalOpen, setIsAddModalOpen] = useState(false);
const [editingCategory, setEditingCategory] = useState(null);
const [confirmDelete, setConfirmDelete] = useState(null);

useEffect(() => {
  fetchCategories();
}, []);

const fetchCategories = async () => {
  try {
    const data = await obtenerCategoriasGasto();
    setCategories(data);
  } catch (error) {
    console.error("Error al cargar categorías:", error);
  }
};

const handleAddCategory = async (nombreCategoria) => {

```

Archivo: ExpensesPage.jsx

Este componente de React representa una página de gestión de gastos donde los usuarios pueden ver una lista de gastos y agregar nuevos gastos mediante un modal interactivo.

Dependencias

- React: Utiliza el hook useState para manejar el estado del componente.
- lucide-react: Biblioteca de iconos que se utiliza para mostrar el icono de agregar (Plus).
- Gastoslist: Componente que muestra la lista de gastos registrados.
- Header, Sidebar: Componentes de la interfaz de usuario que representan el encabezado y la barra lateral de la página.
- AddExpenseModal: Componente de modal para agregar un nuevo gasto.

Estado del Componente

- isOpenModal: Booleano que indica si el modal para agregar un nuevo gasto está abierto.
- refreshExpenses: Booleano utilizado para indicar cuando se debe refrescar la lista de gastos después de que se ha agregado un nuevo gasto.

Funcionalidades Principales

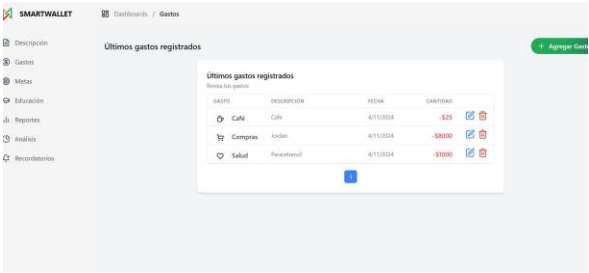
- openModal: Función que establece isOpenModal en true, abriendo el modal.
- closeModal: Función que establece isOpenModal en false, cerrando el modal.
- handleExpenseAdded: Función que se ejecuta cuando se agrega un nuevo gasto, cerrando el modal y actualizando el estado refreshExpenses para refrescar la lista de gastos.

Renderizado Principal

1. Sidebar: Componente de barra lateral para la navegación.
2. Header: Componente de encabezado de la página.
3. Sección Principal:
 - Título: Muestra "Ultimas gastos registrados" como encabezado de la sección de contenido.
 - Botón Agregar Gasto:
 - Muestra un botón estilizado con un icono de Plus y texto de "Agregar Gasto".
 - Al hacer clic, abre el modal de adición de gasto (AddExpenseModal).
4. Gastoslist:
 - Muestra una lista de gastos registrados.
 - Se actualiza cuando refreshExpenses cambia, reflejando nuevos datos cuando se agrega un gasto.
5. AddExpenseModal:
 - Modal para agregar un nuevo gasto.
 - Controlado por isOpenModal para abrir o cerrar.
 - Dispara handleExpenseAdded al agregar un nuevo gasto, refrescando la lista de gastos y cerrando el modal.

Comportamiento del Componente AddExpenseModal

- isOpen: Controla si el modal esta visible o no.
- onClose: Función que se ejecuta al cerrar el modal.
- onExpenseAdded: Función que se ejecuta cuando se confirma la adición de un gasto, provocando un refresco de la lista.



Monto

2000

Categoría

Hogar

Descripción

Tele

Cerrar

Agregar Gasto

Archivo: GoalsManagement.jsx

Este componente de React permite la gestión de categorías de metas, proporcionando una interfaz para visualizar, agregar, editar y eliminar categorías de metas a través de una tabla interactiva y un modal para la adición de nuevas categorías.

Dependencias

- React: Utiliza los hooks `useState` y `useEffect` para manejar el estado y realizar operaciones asíncronas.
- lucide-react: Biblioteca de iconos utilizada para mostrar iconos de acciones como agregar, editar y eliminar (Plus, Pencil, Trash2, Check, X).
- HeaderAdmin, SidebarAdmin: Componentes de interfaz de usuario para el encabezado y la barra lateral de la página de administración.
- Button: Componente de botón personalizado.
- Table, TableBody, TableCell, TableHead, TableHeader, TableRow: Componentes de tabla para mostrar y gestionar las categorías de metas.
- `obtenerCategoriasMeta`, `crearCategoriaMeta`, `actualizarCategoriaMeta`, `eliminarCategoriaMeta`: Funciones importadas de `categoriasApi` para realizar operaciones CRUD en categorías de metas.
- `getIconForCategory`: Función de utilidad para obtener un icono representativo de una categoría.
- AddGoalModal: Componente de modal para agregar una nueva categoría de meta.

Estado del Componente

- `goals`: Arreglo que contiene las categorías de metas obtenidas de la API.
- `isAddModalOpen`: Booleano que indica si el modal de agregar nueva meta está abierto.
- `editingGoal`: Objeto que representa la meta que está siendo editada actualmente.
- `confirmDelete`: ID de la meta que está a punto de ser eliminada para confirmar la acción.

`useEffect`

El `useEffect` se ejecuta al montar el componente, llamando a `fetchGoals` para cargar las categorías de metas desde la API. Maneja errores durante la carga y actualiza el estado `goals` con la respuesta de la API.

Funcionalidades Principales

- `fetchGoals`: Obtiene las categorías de metas desde la API y actualiza el estado `goals`.
- `handleAddGoal`: Agrega una nueva meta llamando a la API y recarga la lista de metas, cerrando el modal al finalizar.
- `startEditing`: Establece la meta que está en edición para permitir la edición en línea.
- `saveGoal`: Guarda los cambios realizados a una meta llamando a la API y recarga la lista de metas.
- `deleteGoal`: Elimina una meta llamando a la API y actualiza el estado `goals` para reflejar la eliminación.
- `setConfirmDelete`: Controla el estado para mostrar opciones de confirmación y cancelación antes de eliminar una meta.

Renderizado Principal

1. SidebarAdmin: Componente de barra lateral para la navegación.
2. HeaderAdmin: Componente de encabezado de la página.
3. Botón Agregar Meta:
 - Abre el modal AddGoalModal para agregar una nueva categoría de meta.
 - Muestra un icono de Plus y un texto de "Agregar Meta".
4. Tabla de Metas:
 - Muestra una lista de categorías de metas con columnas para icono, nombre y acciones.
 - Las acciones incluyen botones para editar (Pencil) y eliminar (Trash2).
5. Edición en Línea:
 - Permite editar el nombre de una categoría de meta directamente en la tabla.
 - Los cambios se confirman con la tecla Enter o se cancelan haciendo clic en un botón (X).

FrontEnd

















- 1. Confirmaci6n de Eliminaci6n:
 - Muestra botones de confirmaci6n (Check) y cancelaci6n (X) cuando se solicita la eliminaci6n de una categor a de meta.
- 1. Modal de Agregar Meta:
 - Se abre al hacer clic en el bot6n de agregar y permite al usuario ingresar un nuevo nombre de meta.
 - Controlado por el estado isAddModalOpen.

Comportamiento del Componente AddGoalModal

- isOpen: Controla si el modal esta visible.
- onClose: Funci6n que se ejecuta al cerrar el modal.
- onSave: Funci6n que se ejecuta al guardar la nueva meta y recargar la lista.

+ Agregar Meta

Gestionar Metas

�CONO	NOMBRE	ACCIONES
�	Hogar	 
�	Salud	 
�	Viaje	 
\$	Finanzas	 
�	Veh�culo	 
�	Educaci�n	 
\$	Ahorro	 
�	Emprendimiento	 

Agregar Nueva Meta

Nombre de la Meta

Cancelar

Guardar

```
export default function GoalsManagement() {
  const [goals, setGoals] = useState([]);
  const [isAddModalOpen, setIsAddModalOpen] = useState(false);
  const [editingGoal, setEditingGoal] = useState(null);
  const [confirmDelete, setConfirmDelete] = useState(null);

  useEffect(() => {
    fetchGoals();
  }, []);

  const fetchGoals = async () => {
    try {
      const data = await obtenerCategoriasMeta();
      setGoals(data);
    } catch (error) {
      console.error("Error al cargar metas:", error);
    }
  };

  const handleAddGoal = async (nombreMeta) => {
    try {
      await crearCategoriaMeta(nombreMeta);
      fetchGoals(); // Volver a cargar las metas despu s de agregar una nueva
      setIsAddModalOpen(false);
    } catch (error) {
      console.error("Error al agregar la meta:", error);
    }
  };
}
```

FrontEnd

Archivo: GoalsPage.jsx

Este componente de React representa una página de gestión de metas donde los usuarios pueden ver, agregar, editar y eliminar metas. La página también incluye la funcionalidad de asociar metas con categorías y se actualiza en tiempo real al realizar cambios.

Dependencias

- React: Utiliza los hooks `useState`, `useEffect`, y `useContext` para manejar el estado, cargar datos y acceder al contexto de autenticación.
- AuthContext: Contexto de autenticación para obtener el token de usuario.
- Header, Sidebar: Componentes de interfaz de usuario que representan el encabezado y la barra lateral de la página.
- AddGoalModal: Componente de modal para agregar una nueva meta.
- Metaslist: Componente que muestra la lista de metas y proporciona funciones para manejar eventos como editar, eliminar y añadir metas.
- `obtenerMetasPorUsuario`, `eliminarMeta`: Funciones importadas de `metasApi` para realizar operaciones CRUD en metas.
- `obtenerCategoriasMeta`: Función importada de `categoriasApi` para obtener categorías de metas.
- Plus: ícono de la biblioteca `lucide-react` para representar la acción de agregar.

Estado del Componente

- `isAddGoalModalOpen`: Booleano que controla la visibilidad del modal de agregar meta.
- `metas`: Arreglo que contiene las metas del usuario.
- `categorías`: Mapa de categorías de metas obtenidas de la API, mapeadas por `categoria_meta_id`.
- `usuarioid`: ID del usuario obtenido a través de la decodificación del token.

Funcionalidades Principales

- `parseJwt`: Función para decodificar el token JWT y obtener el `usuarioid`.
- `handleAddGoalClick`: Abre el modal de agregar meta.
- `handleGoalAdded`: Cierra el modal de agregar meta y actualiza la lista de metas.
- `cargarCategorias`: Obtiene las categorías de metas desde la API y las almacena en el estado `categorías`.
- `cargarMetas`: Obtiene las metas del usuario desde la API, las convierte al formato adecuado y las almacena en el estado `metas`. Solo se ejecuta cuando `categorías` está cargado.
- `handleDeleteMeta`: Elimina una meta por id y recarga la lista de metas.
- `handleEditMeta`: Función de marcador para manejar la edición de metas (aun no implementada).
- `handleAddAmount`: Función de marcador para manejar la adición de monto a una meta (aun no implementada).

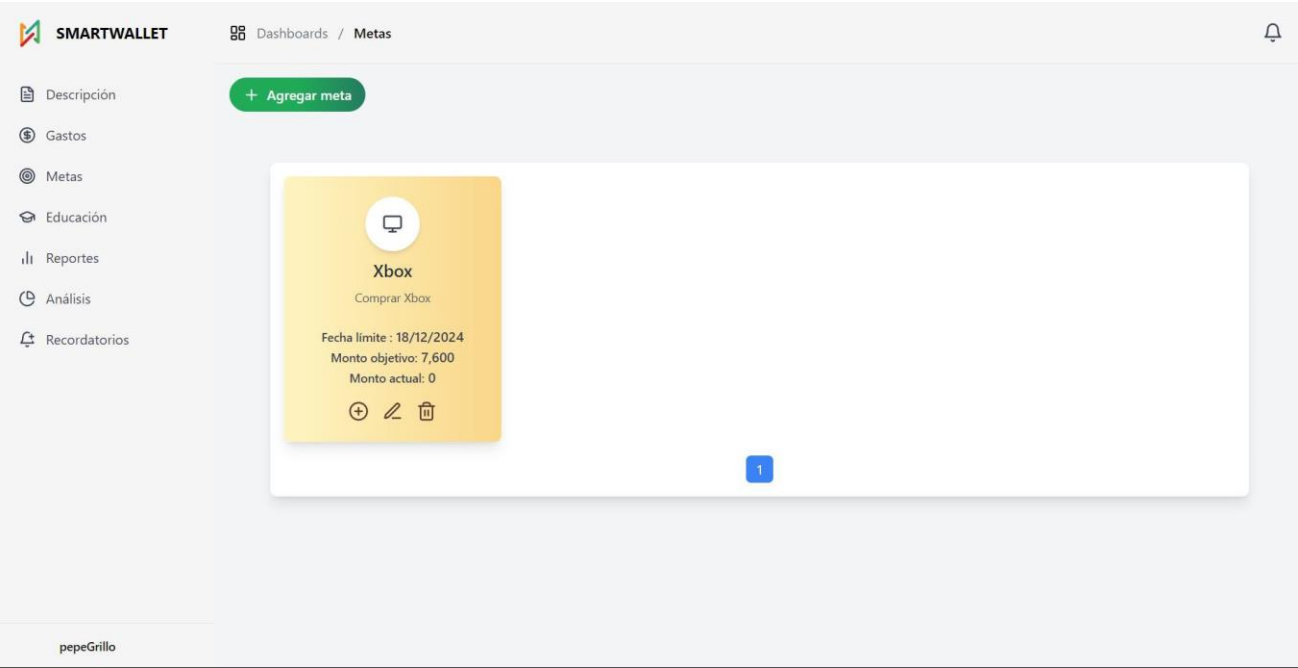
useEffect

- `useEffect` para `cargarCategorias`: Se ejecuta una vez al montar el componente para cargar las categorías.
- `useEffect` para `cargarMetas`: Se ejecuta cada vez que `categorías`, `usuarioid`, o token cambian. Carga las metas solo cuando `categorías` está listo.

FrontEnd

Renderizado Principal

1. Sidebar: Componente de barra lateral para la navegación.
 2. Header: Componente de encabezado de la página.
 3. Botón Agregar Meta:
 - Muestra un botón estilizado con el icono Plus y texto "Agregar meta".
 - Abre el modal de adición de meta al hacer clic.
 4. Sección Principal:
 - Muestra el componente Metaslist con las metas cargadas y funciones para editar, eliminar y añadir montos.
 5. AddGoalModal:
 - Modal que se abre al hacer clic en "Agregar meta".
 - Controlado por el estado isAddGoalModalOpen.
- Comportamiento del Componente AddGoalModal
- isOpen: Controla la visibilidad del modal.
 - onClose: Cierra el modal.
 - onGoalAdded: Llama a handleGoalAdded para cerrar el modal y recargar las metas.



```
6 import { MetasList } from "../componentes/ui/componentes/MetasList";
7 import { obtenerMetasPorUsuario, eliminarMeta } from "../api/metasApi";
8 import { obtenerCategoriasMeta } from "../api/categoriasApi"; // Importar La API de ca
9 import { AuthContext } from "../context/AuthContext";
10
11 export default function GoalsPage() {
12   const [isAddGoalModalOpen, setIsAddGoalModalOpen] = useState(false);
13   const [metas, setMetas] = useState([]); // Estado para almacenar metas
14   const [categorias, setCategorias] = useState(null); // Estado para almacenar categor
15   const { token } = useContext(AuthContext);
16
17   // Función para decodificar el token y obtener el usuarioId
18   const parseJwt = (token) => {
19     try {
20       const base64Url = token.split(".")[1];
21       const base64 = base64Url.replace(/-/g, "+").replace(/_/g, "/");
22       return JSON.parse(window.atob(base64)).id;
23     } catch {}
24     return null;
25   };
26 }
```

FrontEnd

Archivo: HomePage.jsx

Este componente de React representa la página de inicio de una aplicación llamada SmartWallet, que ofrece una vista introductoria a las visitantes con la opción de registrarse y enlaces a las tiendas de aplicaciones. La página presenta un diseño de fondo dinámico con contenido atractivo y botones de acción para captar la atención del usuario.

Dependencias

- **React Router:** Se utilizan Link y useNavigate para la navegación y redirección en la aplicación.
- **Button:** Componente de botón personalizado.
- **Navbar, Footer:** Componentes de interfaz de usuario que representan la barra de navegación superior y el pie de página.
- **Fonda:** Imagen importada para ser utilizada como fondo de la página.

Funcionalidades Principales

- **Navegación:**
 - useNavigate se utiliza para redirigir al usuario a la página de registro al hacer clic en el botón de registro.
- **Enlaces de Descarga:**
 - Se proporcionan enlaces (Link) a las tiendas de aplicaciones (App Store y Google Play) con botones estilizados.

Renderizado Principal

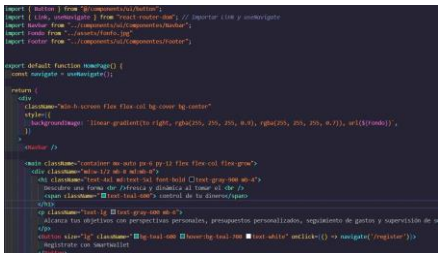
1. **Estilo de Fonda:**
 - Utiliza una imagen de fondo con un gradiente superpuesto para mejorar la visibilidad del texto y otros elementos.
2. **Navbar:**
 - Componente que muestra la barra de navegación superior.
3. **Contenido Principal:**
 - **Encabezado:** Presenta un título destacado y un subtítulo que resaltan las beneficios de SmartWallet.
 - **Botón de Registro:**
 - Un botón grande y estilizado que redirige a la página de registro cuando se hace clic.
 - **Enlaces a App Store y Google Play:**
 - Botones que conducen a las tiendas de aplicaciones para descargar la aplicación, estilizados de manera uniforme y accesible.
4. **Footer:**
 - Componente que aparece al final de la página, asegurando que el pie de página permanezca en la parte inferior, incluso si el contenido de la página es limitado.

Estilización

- **Fonda:** Utiliza backgroundImage con un gradiente y la imagen importada para dar un aspecto moderno y atractivo.
- **Botones:** Estilizados con clases de Tailwind CSS para colores, bordes y efectos de hover.
- **Disposición:**
 - La disposición de los elementos es fluida y se adapta a diferentes tamaños de pantalla, con espaciado y alineación responsiva.

Comportamiento del Componente

- **Button:**
 - Muestra un botón de registro con una función onClick que utiliza navigate('/') para redirigir al usuario a la página de registro.
- **Link:**
 - Se utilizan Link para mostrar botones que, en un futuro, pueden redirigir a las respectivas tiendas de aplicaciones con las URLs adecuadas.



Archivo: HowItWorks.jsx

¿Este componente de React representa la página IC6mo funciona? de la aplicaci6n SmartWallet, que describe sus principales características y beneficios para las usuarios. incluye una introducci6n y una llamada a la acci6n para incentivar a las visitantes a registrarse y comenzar a usar la aplicaci6n.

Dependencias

- React Router: Utiliza useNavigate para redirigir a las usuarios a la página de registro.
- lucide-react: Biblioteca de iconos utilizada para representar visualmente las características de la aplicaci6n.
- Navbar, Footer: Componentes de la interfaz de usuario que representan la barra de navegaci6n superior y el pie de página.

Funcionalidades Principales

- useNavigate: Hook de React Router que permite la redirecci6n programática a otras rutas, en este caso, a la página de registro.
- Muestra de Caracterfsticas: Un arreglo features contiene las características clave de SmartWallet. Cada elemento incluye:
 - icon: icono representativo de la funcionalidad.
 - title: Título de la funcionalidad.
 - description: Descripci6n breve de la funcionalidad.

Renderizado Principal

1. Navbar:
 - Componente que muestra la barra de navegaci6n superior.
2. Secci6n Principal:
 - Título y Descripci6n General:
 - ¿Presenta un título “IC6mo funciona SmartWallet?" y una breve descripci6n sobre c6mo la aplicaci6n ayuda a las usuarios a gestionar sus finanzas.
 - Cuadrícula de Caracterfsticas:
 - Muestra las características de SmartWallet en una cuadrícula responsiva.
 - Cada característica se muestra en una tarjeta con un fondo blanco, icono, título y descripci6n.
 - Llamada a la Acci6n:
 - incluye un mensaje motivacional y un bot6n estilizado que redirige al usuario a la página de registro al hacer clic.
3. Footer:
 - Componente que se muestra al final de la página, proporcionando informaci6n de pie de página.

Estilizaci6n

- Tarjetas de Características:
 - Diseñadas con un fondo blanco y sombra que aparece al pasar el cursor (hover:shadow-lg) para dar un efecto visual atractivo.
 - iconos contenidos en un círculo con fondo de color bg-teal-100.
- Bot6n de Llamada a la Acci6n:
 - Bot6n estilizado con clases de Tailwind CSS para colores y efectos de hover.
 - Redirige a la página de registro al hacer clic.

Comportamiento del Componente

- Bot6n "Comience gratis ahora":
 - Redirige al usuario a la página de registro con navigate('/register') al hacer clic.
- Cuadrícula Responsiva:
 - La cuadrícula de características se adapta a diferentes tamaños de pantalla usando grid-cols-1, md:grid-cols-2, y lg:grid-cols-4 para mostrar las tarjetas de manera flexible.



```
div className="container mx-auto px-4">
  <h2 className="text-3xl font-bold text-center" >text-gray-900 mb-8">¿cómo funciona Smartwallet?</h2>
  <p className="text-center" >text-gray-600 max-w-2xl mx-auto mb-12">
    Smartwallet simplifica su vida financiera al brindarle herramientas poderosas para administrar su dinero de
  </p>
  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
    {features.map((feature, index) => (
      <div key={index} className="bg-white hover:shadow-lg transition-shadow duration-300 p-6 rounded-lg">
        <div className="flex flex-col items-center">
          <div className="rounded-full bg-teal-100 p-3 mb-4">
            {feature.icon}
          </div>
          <h3 className="text-xl font-semibold" >text-gray-900 text-center">{feature.title}</h3>
          <p className="text-gray-600 text-center">{feature.description}</p>
        </div>
      </div>
    ))}
  </div>
  <div className="mt-16 text-center">
    <h3 className="text-2xl font-bold" >text-gray-900 mb-4">¿listo para tomar el control de sus finanzas?</h3>
    <p className="text-gray-600 mb-8">
```


FrontEnd

Archivo: Login.jsx

Este componente de React representa la página de inicio de sesión de la aplicación SmartWallet, proporcionando una interfaz interactiva y animada para que los usuarios puedan autenticarse en la plataforma.

Dependencias

- React: Utiliza los hooks useState y useContext para manejar el estado y acceder al contexto de autenticación.
- AuthContext: Contexto de autenticación que proporciona la función login para guardar el token y el rol del usuario.
- loginUsuario: Función importada de usuariosApi que maneja la solicitud de inicio de sesión.
- framer-motion: Biblioteca para animaciones y transiciones en React.
- react-router-dom: Utiliza Link para enlaces y useNavigate para redirigir a otras páginas.

Estado del Componente

- email: Estado que almacena el valor del campo de correo electrónico.
- password_usuario: Estado que almacena el valor del campo de contraseña.

Funcionalidades Principales

- handlelogin:
 - Función que maneja el inicio de sesión al enviar el formulario.
 - Llama a loginUsuario con email y password_usuario y maneja la respuesta.
 - Si la autenticación es exitosa (token y rol presentes), guarda el token y rol en el contexto de autenticación y redirige al usuario a la página correspondiente según su rol (/admin-overview o /user-overview).
 - Muestra un error en la consola si la autenticación falla.

Animaciones

- framer-motion: Se utiliza para agregar animaciones suaves al cargar el componente y al interactuar con los elementos.
 - fadeIn: Variantes de animación para que los elementos aparezcan con un efecto de desvanecimiento y desplazamiento vertical.

Renderizado Principal

1. Contenedor Principal:
 - Fondo con un degradado de color bg-gradient-to-br from-green-100 to-green-200.
2. Sección de Imagen de Bienvenida:
 - Muestra una imagen de fondo con una superposición de gradiente y un mensaje de bienvenida animado.
3. Formulario de inicio de Sesión:
 - Sección de entrada de datos con campos de email y contraseña, ambos estilizados y con validación requerida.
 - Botón de inicio de sesión estilizado con efectos de hover y focus.
4. Enlaces Complementarios:
 - Registrarse: Redirige a la página de registro.
 - Olvidaste tu contraseña: Enlace a la página de recuperación de contraseña (sin funcionalidad específica en este fragmento).
5. Texto "SMARTWALLET":
 - Texto estilizado con baja opacidad para efecto visual decorativo.



Estilización

- Formulario y Contenedor:
 - Estilizados con clases de Tailwind CSS para espaciado, bordes, sombras, y efectos de hover y focus.
- Botones y Campos de Entrada:
 - Colores y efectos de transición para mejorar la experiencia del usuario.
- Imagen de Fondo:
 - Imagen con object-cover para mantener las proporciones y un gradiente para mejorar la legibilidad del texto superpuesto.

Comportamiento del Componente

- Redirección:
 - Utiliza useNavigate para redirigir a las páginas de /admin-overview o /user-overview tras un inicio de sesión exitoso, basado en el rol del usuario.
- Animaciones:
 - Las animaciones proporcionadas por framer-motion aseguran que los elementos aparezcan de manera atractiva y fluida al cargar la página y al interactuar con los usuarios.

```

4 import { motion } from "framer-motion";
5 import { Link, useNavigate } from "react-router-dom";
6
7 export default function Login() {
8   const { login } = useContext(AuthContext);
9   const [email, setEmail] = useState("");
10  const [password_usuario, setPassword_usuario] = useState("");
11  const navigate = useNavigate();
12
13  const handleLogin = async (e) => {
14    e.preventDefault();
15    const data = await loginUsuario(email, password_usuario);
16
17    if (data.token && data.rol) {
18      // Verifica que el token y el rol están presentes
19      login(data.token, data.rol); // Guarda el token y el rol en el contexto
20
21      if (data.rol === "admin") {
22        navigate("/admin-overview"); // Redirige a la página de administración
23      } else {
24        navigate("/user-overview"); // Redirige a la vista general del usuario
25      }
26
27      console.log("Usuario logeado con éxito");
28    }
29  }
30 }

```

```

<div className="flex flex-col md:flex-row w-full max-w-6xl mx-auto my-8 shadow-2xl rounded-xl overflow-hidden"
  <motion.div
    className="md:w-1/2 bg-cover bg-center relative"
    initial={{ opacity: 0, scale: 0.9 }}
    animate={{ opacity: 1, scale: 1 }}
    transition={{ duration: 0.5 }}
  >
    
    <div className="absolute inset-0 bg-gradient-to-t from-green-200/50 to-transparent" />
    <motion.div
      className="absolute bottom-0 left-0 p-8 text-white"
      initial="hidden"
      animate="visible"
      variants={fadeIn}
      transition={{ delay: 0.2, duration: 0.5 }}
    >
      <h2 className="text-3xl font-bold mb-2">Bienvenido de vuelta</h2>
      <p className="text-lg">
        Continúa tu viaje hacia el éxito financiero con SmartWallet
      </p>
    </motion.div>
  </div>

```

FrontEnd

Archivo: NotificationsPage.jsx

Este componente de React representa la página de notificaciones, donde los usuarios pueden visualizar y administrar sus notificaciones. Ofrece un diseño organizado con una barra lateral y un encabezado superior, proporcionando un espacio principal para la lista de notificaciones.

Dependencias

- **React:** Utiliza el hook `useState` para manejar el estado de actualización de la lista de notificaciones.
- **Notificacioneslist:** Componente que muestra la lista de notificaciones, proporcionando funciones de visualización y eliminación.
- **Header, Sidebar:** Componentes de la interfaz de usuario que representan el encabezado y la barra lateral de la página.

Estado del Componente

- refreshNotifications: Booleano que controla la actualización de la lista de notificaciones. Cambiar este estado desencadena la recarga del componente Notificacioneslist.

Funcionalidades Principales

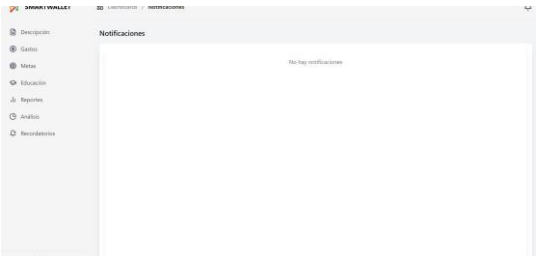
- `handleNotificationDeleted`:
 - Funci6n que se ejecuta cuando se elimina una notificaci6n.
 - Cambia el estado `refreshNotifications` para forzar la recarga del componente `Notificacioneslist` y actualizar la lista de notificaciones.

Renderizado Principal

1. Sidebar:
 - Componente de barra lateral para la navegación.
2. Header:
 - Componente de encabezado de la página.
3. Contenedor de Notificaciones:
 - Contiene un título "Notificaciones" y el componente `Notificacioneslist`.
 - `Notificacioneslist`: Muestra las notificaciones actuales y se refresca cuando `refreshNotifications` cambia.
 - El contenedor está estilizado con un fondo blanco, bordes redondeados y sombras, y permite desplazarse verticalmente si el contenido supera la altura disponible (`overflow-y-auto`).

Comportamiento del Componente

- Actualización Dinámica:
 - La función `handleNotificationDeleted` cambia el estado `refreshNotifications` al eliminar una notificación, lo que permite actualizar la lista sin necesidad de recargar la página.
- Scroll Vertical:
 - El contenedor de `Notificacioneslist` permite el desplazamiento vertical para manejar listas largas de notificaciones.



```

    useState() from "react";
    NotificationsList from "../components/ui/componentes/NotificacionesList"; // Componente a
    Header from "../components/ui/componentes/Header";
    Sidebar from "../components/ui/componentes/Sidebar";

    default function NotificationsPage() {
    t [refreshNotifications, setRefreshNotifications] = useState(false); // Controla La actualizaci
    función para manejar la eliminación de una notificación y refrescar la lista
    t handleNotificationDeleted = () => {
    t refreshNotifications(refreshNotifications); // Refresca NotificacionesList

    rn {
    div className="flex h-screen">
    /* Sidebar */
    <div className="w-64 bg-[#F9F5F5]">
    <Sidebar />
    </div>

    /* Contenido principal */
    <div className="flex-1 flex flex-col">
    /* Header */
    <Header />

    /* Contenido de Notificaciones */

```

FrontEnd

Archivo: PricingPage.jsx

Este componente de React representa la página de precios de la aplicación SmartWallet, donde los usuarios pueden ver y seleccionar diferentes planes de suscripción. Cada plan muestra características específicas y permite a los usuarios iniciar el proceso de registro o abrir un modal de pago para elegir un plan.

Dependencias

- React: Utiliza el hook useState para manejar el estado de la vista de precios y el estado del modal de pago.
- PropTypes: Verifica los tipos de datos de las props pasadas al componente PricingTier.
- react-router-dom: Utiliza useNavigate para redirigir a la página de registro.
- lucide-react: Biblioteca de iconos utilizada para mostrar iconos de características incluidas (Check) o no incluidas (X).
- Navbar, Footer: Componentes de interfaz de usuario que representan la barra de navegación y el pie de página.
- PaymentModal: Componente de modal que permite al usuario ver detalles de pago y proceder con la selección de un plan.

Estado del Componente

- isAnnual: Booleano que indica si los precios mostrados son anuales o mensuales.
- isModalOpen: Booleano que controla la visibilidad del PaymentModal.
- selectedPackage: Objeto que almacena la información del paquete seleccionado para mostrar en el PaymentModal.

Funcionalidades Principales

- openModal:
 - Función que se ejecuta al seleccionar un plan (diferente al gratuito).
 - Establece el selectedPackage con la información del plan y abre el modal de pago.
- closeModal:
 - Función que cierra el PaymentModal.

PricingTier Componente

- Props:
 - name: Nombre del plan.
 - price: Precio del plan (mensual o anual).
 - features: Lista de características del plan, donde cada una tiene un texto y un indicador de si está incluida.
 - recommended: Booleano que indica si el plan es recomendado.
 - onSelect: Función de callback para manejar la selección del plan.
- Funcionalidad:
 - Redirige al usuario a la página de registro si el plan es gratuito, de lo contrario, ejecuta onSelect para abrir el modal de pago.

```
import PropTypes from 'prop-types';
import { useNavigate } from 'react-router-dom';
import Navbar from '../components/ui/Componentes/Navbar';
import PaymentModal from '../components/ui/Componentes/Modales/PaymentModal';
import Footer from '../components/ui/Componentes/Footer';

const PricingTier = ({ name, price, features, recommended, onSelect }) => {
  const navigate = useNavigate();

  const handleClick = () => {
    if (name === 'Gratis') {
      navigate('/register');
    } else {
      onSelect();
    }
  };

  return (
    <div className={`flex flex-col p-4 bg-white rounded-lg shadow-md ${recommended ? 'border-2 border-green-500' : ''}`>
      <h3 className="text-2xl font-bold text-gray-900">{name}</h3>
      <div className="mt-4 text-green-600 text-5xl font-bold">
        {price}
        <span className="text-xl text-gray-500">/mo</span>
      </div>
      <ul className="mt-6 space-y-4 flex-grow">
        {features.map((feature, index) => (
          <li key={index} className="flex items-center">
```

Renderizado Principal

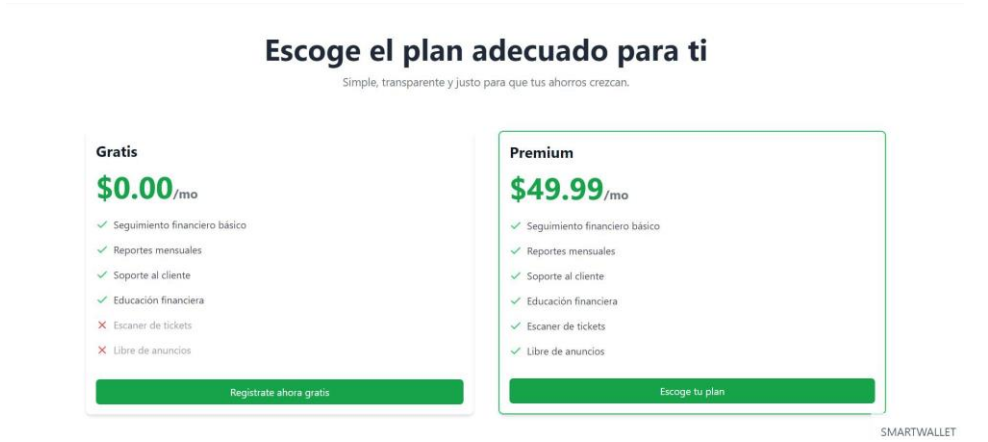
- 1. Navbar:
 - Componente de barra de navegación.
- 2. Sección de Precios:
 - Título principal con una breve descripción.
 - Cuadrícula de planes de precios:
 - Muestra cada plan con el componente PricingTier.
 - El plan recomendado se resalta con un borde de color.
- 3. Modal de Pago:
 - Controlado por el estado isModalOpen.
 - Muestra el nombre y el precio del paquete seleccionado.
- 4. Footer:
 - Componente de pie de página.

Estilización

- Componente PricingTier:
 - Fondo blanco, bordes redondeados y sombra (shadow-md).
 - Un borde verde para destacar el plan recomendado.
 - Botón estilizado para iniciar la acción de registro o selección de plan.
- iconos de Características:
 - iconos de verificación (Check) en verde y de exclusión (X) en rojo para indicar si una característica está incluida.

Comportamiento del Componente

- Cambia de Plan:
 - Permite alternar entre precios anuales y mensuales, mostrando el precio adecuado según el estado isAnnual.
- Selección de Plan:
 - Si el plan es gratuito, el botón redirige a la página de registro usando navigate('/register').
 - Si el plan no es gratuito, se abre un PaymentModal con detalles para proceder con el pago.



Archivo: Register.jsx

Este componente de React representa la página de registro para nuevos usuarios de la aplicación SmartWallet, proporcionando una interfaz visualmente atractiva con validación de formularios y animaciones.

Dependencias

- React: Utiliza el hook useState para manejar el estado de los campos de entrada y el manejo de errores.
- react-router-dom: Utiliza useNavigate para redirigir al usuario a la página de inicio de sesión y Link para proporcionar enlaces internos.
- framer-motion: Biblioteca de animación que permite transiciones suaves de entrada y salida.
- registerUsuario: Función importada de usuariosApi para manejar la solicitud de registro de usuario.
- Logo: importa el logo de la aplicación desde la carpeta de assets.

Estado del Componente

- name: Almacena el valor del campo de nombre.
- email: Almacena el valor del campo de correo electrónico.
- password: Almacena el valor del campo de contraseña.
- confirmPassword: Almacena el valor del campo de confirmación de contraseña.
- error: Almacena mensajes de error para la validación del formulario.

Funcionalidades Principales

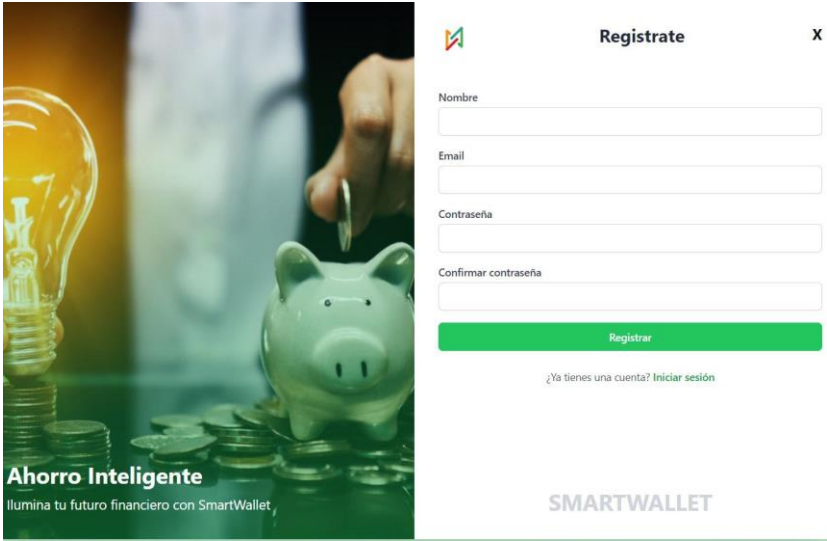
- handleSubmit:
 - Función que se ejecuta al enviar el formulario.
 - Verifica si password y confirmPassword coinciden. Si no coinciden, muestra un mensaje de error.
 - Llama a la función registerUsuario para registrar al usuario.
 - Si el registro es exitoso, redirige al usuario a la página de inicio de sesión (/login).
 - Muestra un mensaje de error si hay algún problema durante el registro.

Animaciones

- framer-motion: Se utiliza para aplicar transiciones suaves en la carga de elementos del componente.
 - fadeIn: Variantes de animación para que los elementos aparezcan con un efecto de desvanecimiento y desplazamiento vertical.

Renderizado Principal

1. Sección de Imagen de Introducción:
 - Muestra una imagen de fondo con un gradiente superpuesto y un mensaje de bienvenida animado.



FrontEnd

1. Formulario de Registro:

- Contiene campos para Nombre, Email, Contraseña y Confirmar Contraseña.
- Bot6n de envío estilizado para registrar al usuario.

1. Enlaces Complementarios:

- Enlace para volver a la página de inicio (icono "X").
- Enlace para redirigir a la página de inicio de sesi6n si el usuario ya tiene una cuenta.

1. Mensajes de Error:

- Muestra un mensaje de error si las contraseñas no coinciden o si hay alg6n problema al registrar al usuario.

1. Texto "SMARTWALLET":

- Texto decorativo con baja opacidad al final de la secci6n de formulario.

Estilizaci6n

- Formulario y Contenedor:
- Utiliza Tailwind CSS para el diseo y la estilizaci6n, con clases para bordes, sombras y efectos de hover y focus.
- Bot6n de Registro:
- Colores y efectos de transici6n para mejorar la experiencia del usuario.
- Imagen de Fonda:
- Imagen con object-cover para mantener las proporciones, combinada con un gradiente para mejorar la visibilidad del texto.

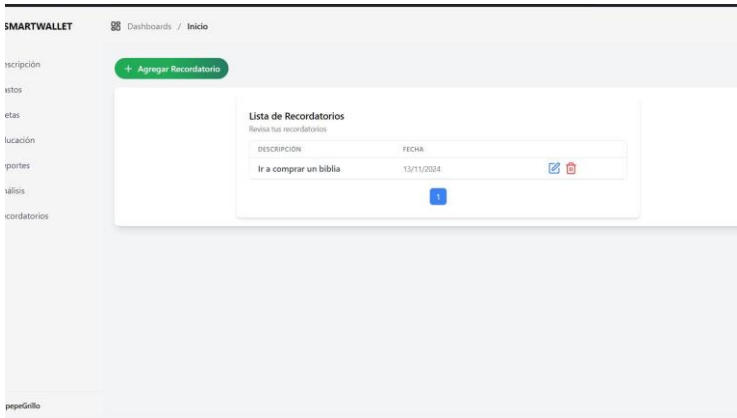
Comportamiento del Componente

- Validaci6n de Contraseña:
- Verifica que la contraseña y la confirmaci6n de contraseña coincidan antes de enviar el formulario.
- Redirecci6n:
- Utiliza useNavigate para redirigir al usuario a la p6gina de inicio de sesi6n despu6s de un registro exitoso.
- Mensajes de Error:
- Muestra mensajes de error claros en caso de problemas de validaci6n o fallos de registro.

```
export default function Register() {  
  return (  
    <div className="flex min-h-screen bg-gradient-to-br from-green-100 to-green-200">  
      <div className="flex flex-col md:flex-row w-full max-w-6xl mx-auto my-8 shadow-2xl rounded-xl overflow-hidden">  
        <motion.div  
          className="md:w-1/2 bg-cover bg-center relative"  
          initial={{ opacity: 0, scale: 0.9 }}  
          animate={{ opacity: 1, scale: 1 }}  
          transition={{ duration: 0.5 }}  
        >  
            
          <div className="absolute inset-0 bg-gradient-to-t from-green-600/50 to-transparent" />  
          <motion.div  
            className="absolute bottom-0 left-0 p-8 text-white"  
            initial="hidden"  
            animate="visible"  
            variants={fadeIn}  
            transition={{ delay: 0.2, duration: 0.5 }}  
          >  
            <h2 className="text-3xl font-bold mb-2">Ahorro Inteligente</h2>  
            <p className="text-lg">  
              Ilumina tu futuro financiero con SmartWallet  
            </p>  
          </motion.div>  
        </motion.div>  
      </div>  
    </motion.div>  
  )  
}
```


FrontEnd

- 1. Archivo: RemindersPage.jsx
- 2. Este componente de React representa la página de recordatorios, donde las usuarios pueden ver una lista de recordatorios existentes y agregar nuevos recordatorios a través de un modal interactivo.
- 3. Dependencias
 - React: Utiliza el hook useState para manejar el estado de visibilidad del modal y el control de la actualización de la lista de recordatorios.
 - lucide-react: Biblioteca de iconos utilizada para mostrar un icono de agregar (Plus).
 - Recordatorioslist: Componente que muestra la lista de recordatorios.
 - Header, Sidebar: Componentes de interfaz de usuario que representan el encabezado y la barra lateral de la página.
 - AddReminderModal: Componente de modal que permite agregar un nuevo recordatorio.
- 4. Estado del Componente
 - isModalOpen: Booleano que controla la visibilidad del AddReminderModal.
 - refreshReminders: Booleano que controla la actualización de la lista de recordatorios. Cambiar este estado desencadena la recarga de Recordatorioslist.
- 5. Funcionalidades Principales
 - openModal:
 - Función que abre el AddReminderModal al establecer isModalOpen en true.
 - closeModal:
 - Función que cierra el AddReminderModal al establecer isModalOpen en false.
 - handleReminderAdded:
 - Función que se ejecuta después de agregar un nuevo recordatorio. Cierra el modal y actualiza el estado refreshReminders para forzar la recarga de Recordatorioslist.
- 6. Renderizado Principal
 - a. Sidebar:
 - Componente de barra lateral para la navegación.
 - b. Header:
 - Componente de encabezado de la página.
 - c. Botón "Agregar Recordatorio":
 - Botón estilizado con un icono de Plus y texto "Agregar Recordatorio".
 - Al hacer clic, abre el modal de adición de recordatorios.
 - d. Contenedor de Recordatorios:
 - Contiene el componente Recordatorioslist, que muestra la lista de recordatorios y se actualiza cuando refreshReminders cambia.
 - Estilizado con un fondo blanco, bordes redondeados y sombras (shadow-lg).
 - e. AddReminderModal:
 - Modal que se abre para permitir al usuario ingresar un nuevo recordatorio.
 - Controlado por el estado isModalOpen y ejecuta handleReminderAdded al confirmar la adición.



7. Estilización

Botón de Agregar:

Utiliza clases de Tailwind CSS para un diseño atractivo y efectos de hover (bg-gradient-to-r con un gradiente verde).

Contenedor Principal:

Diseñado con clases de Tailwind CSS para espaciado, bordes y sombras, manteniendo una apariencia moderna y clara.

Comportamiento del Componente

Actualización Dinámica:

La función `handleReminderAdded` cambia el estado `refreshReminders` después de agregar un nuevo recordatorio, lo que permite que `Recordatorioslist` se recargue automáticamente y muestre la lista actualizada.

Modal de Adición:

Controla la visibilidad y se cierra automáticamente después de agregar un recordatorio exitosamente.

Archivo: ReportsPage.jsx

Este componente de React representa la página de reportes de la aplicación SmartWallet, donde los usuarios pueden visualizar, buscar, agregar y eliminar reportes. La página incluye un sistema de búsqueda para filtrar reportes y modales para añadir y confirmar la eliminación de reportes.

Dependencias

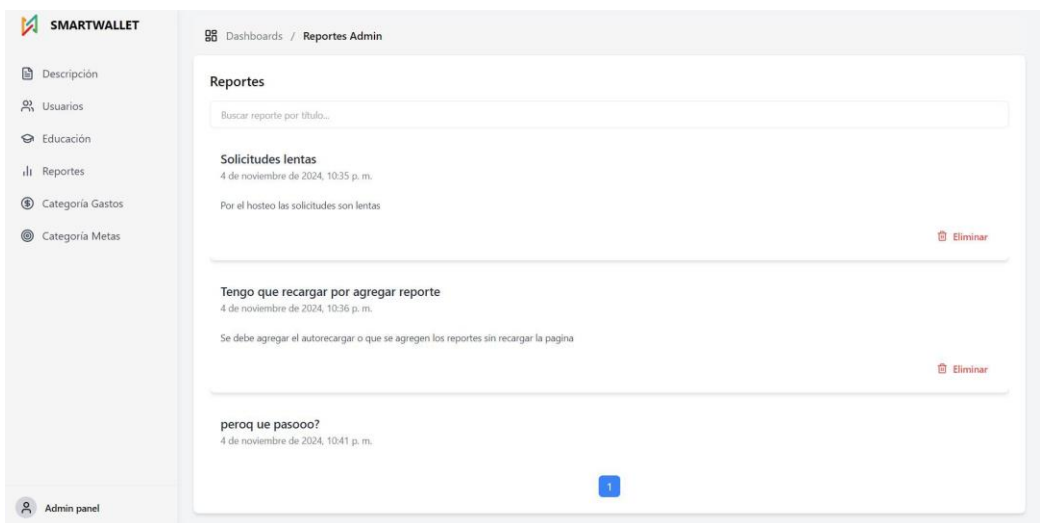
- **React:** Utiliza los hooks `useState`, `useEffect`, y `useContext` para manejar el estado y la carga de datos.
- **AuthContext:** Contexto de autenticación que proporciona el token del usuario.
- **Button:** Componente de botón estilizado.
- **Lucide-react:** Biblioteca de iconos utilizada para mostrar el icono de `PlusCircle`.
- **ScrollArea:** Componente que permite mostrar listas largas con desplazamiento.
- **Header, Sidebar:** Componentes de interfaz de usuario para la estructura de la página.
- **CardReport:** Componente que muestra cada reporte.
- **AddReportModal:** Componente de modal que permite agregar un nuevo reporte.
- **obtenerReportesPorUsuario, crearReporte, eliminarReporte:** Funciones importadas de `reportesApi` para realizar operaciones CRUD en reportes.

Estado del Componente

- **isAddReportModalOpen:** Booleano que controla la visibilidad del `AddReportModal`.
- **isDeleteModalOpen:** Booleano que controla la visibilidad del modal de confirmación de eliminación.
- **reportes:** Arreglo que almacena todos los reportes del usuario.
- **filteredReportes:** Arreglo que almacena los reportes filtrados por el término de búsqueda.
- **searchTerm:** Cadena que representa el término de búsqueda.
- **reporteAEliminar:** Almacena el ID del reporte que se desea eliminar.

Funcionalidades Principales

- **parseJwt:**
 - Función que decodifica el token JWT para obtener el `usuario_id`.
- **cargarReportes:**
 - Función que obtiene los reportes del usuario y los almacena en `reportes` y `filteredReportes`.
- **handleAddReportClick:**
 - Abre el `AddReportModal`.
- **handleReportAdded:**
 - Función que se ejecuta después de agregar un reporte. Vuelve a cargar los reportes y cierra el modal.
- **handleDelete:**
 - Función que establece el ID del reporte a eliminar y abre el modal de confirmación de eliminación.



FrontEnd

- confirmDeleteReport:
- Elimina el reporte con el ID almacenado en reporteAEliminar y vuelve a cargar los reportes.
- handleSearchChange:
- Filtra los reportes basándose en el término de búsqueda y actualiza filteredReportes.

Renderizado Principal

1. Sidebar:

- Componente de barra lateral para la navegación.

1. Header:

- Componente de encabezado de la página.

1. Sección Principal:

- Título "Mis reportes" y un botón para abrir el AddReportModal.
- Campo de búsqueda para filtrar reportes por título.
- Lista de reportes mostrada en el componente ScrollArea, que incluye el componente CardReport para cada reporte.

1. AddReportModal:

- Modal que se abre para agregar un nuevo reporte y se cierra al confirmar la adición.

1. Modal de Confirmación de Eliminación:

- Modal que se muestra cuando el usuario confirma la eliminación de un reporte, con opciones de cancelar o confirmar la acción.

Estilización

- Botón de Agregar Reporte:
- Utiliza clases de Tailwind CSS para un diseño atractivo y efectos de hover (bg-red-500, hover:bg-red-600).
- Campo de Búsqueda:
- Campo de entrada estilizado con bordes redondeados y sombreado.
- Contenedor de Reportes:
- Utiliza bg-white, p-4, y rounded-lg para un diseño limpio y moderno.

Comportamiento del Componente

- Actualización Dinámica:
- La función handleReportAdded y confirmDeleteReport actualizan la lista de reportes al agregar o eliminar un reporte.
- Búsqueda en Tiempo Real:
- La función handleSearchChange filtra los reportes a medida que el usuario escribe en el campo de búsqueda.

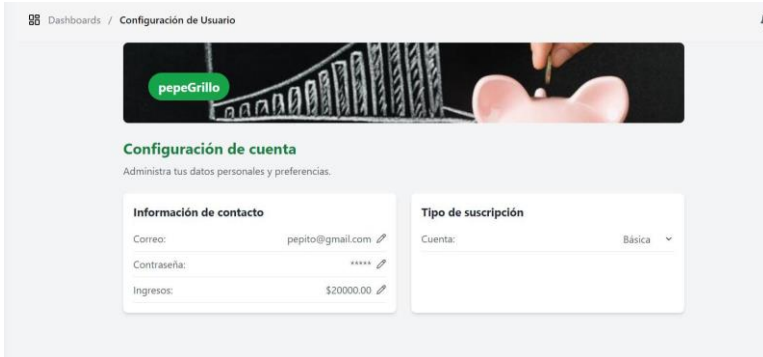
```
6 import Sidebar from "../components/ui/Componentes/Sidebar";
7 import CardReport from "../components/ui/Componentes/CardReport";
8 import AddReportModal from "../components/ui/Componentes/Modales/AddReportModal";
9 import { AuthContext } from "../context/AuthContext";
0 import { obtenerReportesPorUsuario, crearReporte, eliminarReporte } from "../api/reportesApi";
1
2 export default function ReportsPage() {
3   const { token } = useContext(AuthContext);
4   const [isAddReportModalOpen, setIsAddReportModalOpen] = useState(false);
5   const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
6   const [reportes, setReportes] = useState([]);
7   const [filteredReportes, setFilteredReportes] = useState([]); // Para almacenar los reportes filtrados
8   const [searchTerm, setSearchTerm] = useState(""); // Estado para el término de búsqueda
9   const [reporteAEliminar, setReporteAEliminar] = useState(null);
10
11   // Obtener usuario_id del token
12   const parseJwt = (token) => {
13     try {
14       const base64Url = token.split(".")[1];
15       const base64 = base64Url.replace(/-/g, "+").replace(/_/g, "/");
16       return JSON.parse(window.atob(base64));
17     } catch {
```

FrontEnd

- Archivo: UserConfigurations.jsx
- Este componente de React representa la página de configuración de usuario de la aplicación SmartWallet, permitiendo a los usuarios ver actualizar su perfil, datos de contacto y preferencias de suscripción.
- Dependencias
 - React: Utiliza los hooks useState, useEffect, y useContext para manejar el estado y las operaciones de carga de datos.
 - AuthContext: Contexto de autenticación que proporciona el token del usuario.
 - Sidebar, Header, UserProfileCard, InformationCard, AccountTypeSelect: Componentes de la interfaz de usuario que organizan la página de configuración.
 - EditModal, PaymentModal: Componentes de modal que permiten la edición de la información del usuario y la gestión de pagos para cambiar de suscripción.
 - obtenerUsuarios, actualizarUsuario: Funciones importadas de usuariosApi para obtener y actualizar los datos del usuario.
- Estado del Componente
 - isEditModalOpen: Booleano que controla la visibilidad del EditModal.
 - isPaymentModalOpen: Booleano que controla la visibilidad del PaymentModal.
 - selectedSection: Estado que guarda la sección actual que se está editando.
 - editValue: Estado que almacena el valor inicial para el modal de edición.
 - userProfile: Objeto que almacena los datos del perfil del usuario, incluyendo nombre, email, ingresos y tipo de suscripción.
- Funcionalidades Principales
 - parseJwt:
 - Función que decodifica el token JWT para obtener el usuarioId.
 - useEffect:
 - Carga los datos del usuario al montar el componente.
 - fetchUserData:
 - Función asíncrona que obtiene los datos del usuario de la API y actualiza el estado userProfile.
 - openEditModal:
 - Abre el modal de edición y establece la sección seleccionada y el valor inicial.
 - handleAccountTypeChange:
 - Maneja el cambio de tipo de cuenta. Si el usuario selecciona "Premium", abre el PaymentModal.
 - handleSaveChanges:
 - Función que actualiza los datos del usuario y cierra el modal de edición.
- Renderizado Principal
 - a. Sidebar:
 - Componente de barra lateral para la navegación.
 - b. Header:
 - Componente de encabezado de la página.
 - c. UserProfileCard:
 - Muestra la tarjeta de perfil del usuario con su nombre, imagen de perfil y de fondo, y un botón para editar.
 - d. Sección "Configuración de cuenta":
 - Contiene información de contacto y tipo de suscripción.
 - InformationCard para la información de contacto con opción de editar.
 - AccountTypeSelect dentro de una InformationCard para cambiar el tipo de cuenta.
 - e. Modales:
 - EditModal: Permite al usuario editar secciones de su perfil, como el nombre o el email.
 - PaymentModal: Se abre al seleccionar una suscripción Premium y permite al usuario realizar un pago.
- Estilización
 - Componentes y Diseño:
 - Utiliza Tailwind CSS para la disposición, espaciado, bordes redondeados y sombras.
 - Modales:
 - Estilizados para que se adapten al diseño de la página y proporcionen una experiencia de usuario intuitiva.

FrontEnd

- Comportamiento del Componente
 - Carga de Datos:
 - Al cargar la página, se ejecuta `fetchUserData` si `usuarioid` está disponible, para poblar los datos del perfil.
 - Actualización Dinámica:
 - Los datos del perfil se actualizan al guardar cambios en el `EditModal`.
 - Gestión de Suscripción:
 - Si el usuario cambia a una suscripción Premium, se muestra el `PaymentModal` con el precio y la opción de guardar la tarjeta.



```
};

export default function UserConfigurations() {
  const { token } = useContext(AuthContext);
  const [isEditModalOpen, setEditModalOpen] = useState(false);
  const [isPaymentModalOpen, setPaymentModalOpen] = useState(false);
  const [selectedSection, setSelectedSection] = useState(null);
  const [editValue, setEditValue] = useState("");
  const [userProfile, setUserProfile] = useState({
    nombre: "Cargando...",
    email: "",
    password_usuario: "*****",
    ingresos: 0,
    preferencias: {
      cuenta: "Básica",
    },
  });
}
```


FrontEnd

Archivo: UserManagementPage.jsx

Este componente de React representa la página de gestión de usuarios para administradores de la aplicación SmartWallet, permitiendo ver, buscar y eliminar usuarios de la plataforma.

Dependencias

- React: Utiliza los hooks useState y useEffect para manejar el estado y la carga de datos.
- SidebarAdmin, HeaderAdmin: Componentes de la interfaz de usuario que representan la barra lateral y el encabezado de la página de administración.
- Table, TableBody, TableCell, TableHead, TableHeader, TableRow: Componentes de tabla para mostrar la lista de usuarios.
- Button: Componente de botón estilizado.
- lucide-react: Biblioteca de iconos utilizada para mostrar iconos (Trash2, ChevronDown).
- obtenerUsuarios, eliminarUsuario: Funciones importadas de usuariosApi para obtener y eliminar usuarios.

Estado del Componente

- users: Arreglo que almacena todos los usuarios.
- filteredUsers: Arreglo que almacena los usuarios filtrados por el término de búsqueda.
- loading: Booleano que indica si los datos se están cargando.
- showConfirm: Booleano que controla la visibilidad del modal de confirmación de eliminación.
- selectedUser: Almacena el usuario seleccionado para eliminar.
- currentPage: Número de la página actual para la paginación.
- searchQuery: Cadena que representa el término de búsqueda.

Funcionalidades Principales

- useEffect:
 - Llama a fetchUsers al montar el componente para cargar la lista de usuarios.
- fetchUsers:
 - Función asíncrona que obtiene la lista de usuarios desde la API y actualiza el estado users y filteredUsers.
- handleDeleteClick:
 - Abre el modal de confirmación de eliminación y establece el usuario seleccionado.
- confirmDelete:
 - Elimina el usuario seleccionado de la API, actualiza el estado y cierra el modal.
- cancelDelete:
 - Cierra el modal de confirmación de eliminación sin realizar cambios.
- handleSearchChange:
 - Filtra los usuarios basándose en el término de búsqueda y actualiza filteredUsers.
- paginate:
 - Cambia la página actual para mostrar diferentes usuarios en la tabla.

Renderizado Principal

7. SidebarAdmin:

- Componente de barra lateral para la navegación.

8. HeaderAdmin:

- Componente de encabezado de la página.

9. Campo de Búsqueda:

- Input para buscar usuarios por nombre con actualización en tiempo real.

7. Tabla de Usuarios:

- Muestra la lista de usuarios con columnas para el nombre de usuario, fecha de registro, estado (tipo de suscripción) y acciones (botón de eliminar).
- La columna de estado usa clases de Tailwind CSS para mostrar el estado en colores distintivos.

7. Paginación:

- Botones de paginación para navegar entre diferentes páginas de usuarios.

7. Modal de Confirmación de Eliminación:

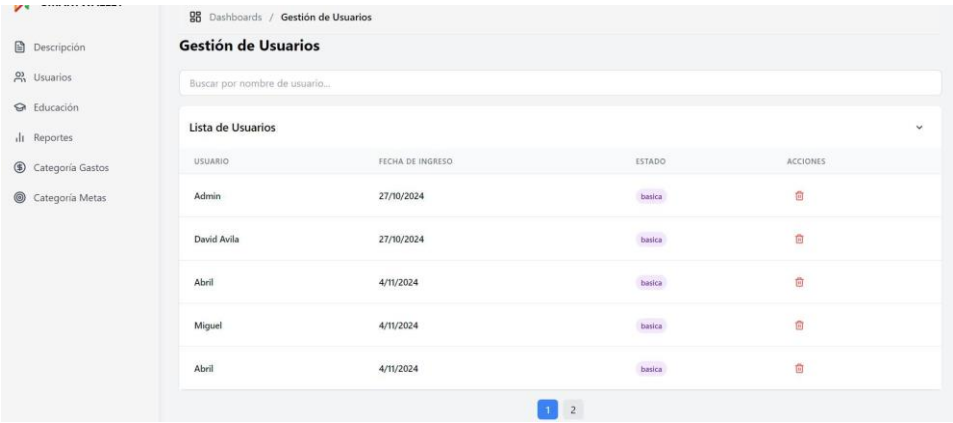
- Modal que se muestra al confirmar la eliminación de un usuario, con opciones para cancelar o confirmar la acción.

Estilización

- Tabla:
- Utiliza clases de Tailwind CSS para el diseño, con bordes redondeados y sombreado (shadow-lg).
- Campo de Búsqueda:
- Input estilizado con bordes y efectos de enfoque (focus:ring-2, focus:ring-blue-500).
- Botones:
- Botones estilizados para acciones de eliminar y de paginación, con colores distintivos para indicar la página actual.

Comportamiento del Componente

- Carga de Datos:
- Carga la lista de usuarios al montar el componente y actualiza la lista al eliminar un usuario.
- Búsqueda Dinámica:
- Filtra los usuarios según el termino de búsqueda ingresado por el usuario.
- Paginación:
- Permite navegar entre diferentes páginas de la lista de usuarios.
- Eliminación de Usuarios:
- Muestra un modal de confirmación antes de eliminar un usuario y actualiza la lista al confirmar la eliminación.



```
// Paginación
const [currentPage, setCurrentPage] = useState(1);
const itemsPerPage = 5;

// Estado para el campo de búsqueda
const [searchQuery, setSearchQuery] = useState("");

useEffect(() => {
  const fetchUsers = async () => {
    try {
      const usuarios = await obtenerUsuarios();
      setUsers(usuarios);
      setFilteredUsers(usuarios); // Inicialmente, todos los usuarios están en filteredUsers
    } catch (error) {
      console.error("Error al cargar los usuarios:", error);
    } finally {
      setLoading(false);
    }
  }
  fetchUsers();
});
```

FrontEnd

7. Archivo: UserOverviewPage.jsx

8. Este componente de React representa la página de resumen del usuario en la aplicación SmartWallet, donde los usuarios pueden visualizar gráficos de gastos, su ingreso mensual y una lista de sus gastos recientes.

9. Dependencias

- React: Utiliza los hooks useState, useEffect, y useContext para manejar el estado y la carga de datos.
- AuthContext: Contexto de autenticación que proporciona el token del usuario.
- Sidebar, Header: Componentes de la interfaz de usuario que representan la barra lateral y el encabezado.
- CategoryExpensesChart, ExpensesDistributionChart, RecentExpensesTable: Componentes para la visualización de gráficos y tablas de gastos.
- obtenerIngresoUsuario, actualizarIngreso: Funciones de usuariosApi para obtener y actualizar el ingreso del usuario.
- obtenerGastosPorUsuario: Función de gastosApi para obtener los gastos del usuario.

10. Estado del Componente

- isIncomeModalOpen: Booleano que controla la visibilidad del modal para ingresar el monto de ingresos.
- income: Valor que almacena el ingreso mensual del usuario.
- expenseData: Datos utilizados para el gráfico de distribución de gastos.
- expenseCategories: Datos utilizados para el gráfico de categorías de gastos.
- recentExpenses: Arreglo que contiene los gastos más recientes del usuario.

11. Funcionalidades Principales

- parseJwt:
 - Función que decodifica el token JWT para obtener el usuario_id.
- useEffect:
 - Llama a las funciones fetchIncome y fetchExpenses al montar el componente para cargar el ingreso y los gastos del usuario.
- fetchIncome:
 - Obtiene el ingreso mensual del usuario. Si el ingreso es nulo o menor o igual a cero, abre el modal de ingreso.
- fetchExpenses:
 - Obtiene los gastos del usuario, los ordena por fecha y los agrupa por categoría para mostrar en gráficos.
- handleSaveIncome:
 - Actualiza el ingreso del usuario al confirmar el nuevo valor ingresado y cierra el modal.

12. Renderizado Principal

g. Sidebar:

- Componente de barra lateral para la navegación.

h. Header:

- Componente de encabezado de la página.

i. Sección de Resumen:

- Contiene gráficos de Gastos por Categoría y Distribución de Gastos, ambos dentro de tarjetas (div) con fondo blanco, bordes redondeados y sombras.

j. Tabla de Gastos Recientes:

- Muestra los últimos tres gastos registrados del usuario.

k. Modal de ingreso:

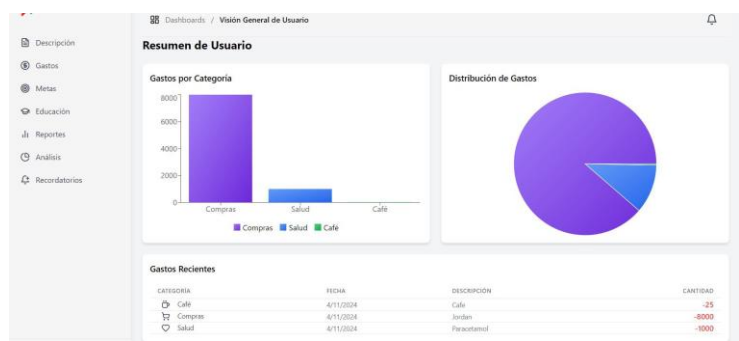
- Modal que se abre cuando no se ha configurado el ingreso mensual, permitiendo al usuario ingresarlo y guardarlo.

13. Estilización

- Tarjetas de Gráficos y Tabla:
 - Utiliza clases de Tailwind CSS (bg-white, shadow-md, rounded-lg) para un diseño limpio y moderno.

FrontEnd

- Modal de ingreso:
 - Modal con fondo oscuro semitransparente (bg-gray-800 bg-opacity-50) y una tarjeta central con sombreado para ingresar el valor de ingresos.
- 7. Comportamiento del Componente
 - Carga de Datos:
 - Al cargar la página, fetchIncome y fetchExpenses obtienen la información necesaria y la actualizan en los estados correspondientes.
 - Gráficos y Visualizaciones:
 - Los datos de gastos se procesan y se muestran en gráficos y una tabla de manera ordenada y visualmente atractiva.
 - ingreso de Usuario:
 - Si el ingreso no está configurado o es cero, el modal se muestra automáticamente para solicitar al usuario que ingrese su ingreso mensual.
 - Actualización de ingresos:
 - Permite al usuario ingresar y guardar su ingreso mensual, cerrando el modal al confirmar la acción.



Configura tu Ingreso Mensual

Establecer tu ingreso mensual permite a la aplicación generar comparaciones de gastos y visualizar gráficas personalizadas sobre tu situación financiera.

Guardar

Archivo: App.jsx

Este archivo representa el componente principal de la aplicación SmartWallet y define las rutas de la aplicación utilizando React Router. También incluye la lógica para manejar la autenticación y proteger ciertas rutas basadas en roles de usuario.

Dependencias

- react-router-dom: Biblioteca utilizada para la navegación y el manejo de rutas.
- AuthProvider: Contexto de autenticación que envuelve la aplicación para proporcionar el estado de autenticación globalmente.
- NotificationProvider, GlobalNotification: Proveedores y componentes para manejar las notificaciones globales en la aplicación.
- ProtectedRoute: Componente que protege rutas y permite el acceso solo a usuarios autenticados con roles específicos.

Rutas de la Aplicación

1. Rutas Publicas:

- /: Página de inicio (HomePage).
- /login: Página de inicio de sesión (Login).
- /register: Pagina de registro (Register).
- /how-it-works: Pagina de cómo funciona la aplicación (HowItWorks).
- /prices: Pagina de precios y suscripciones (PricingTier).
- /about-us: Sección "Sabre Nosotros" (AboutSection).

2. Rutas Protegidas para Usuarios (Role: "usuario"):

- /goals: Pagina de metas del usuario (GoalsPage).
- /analysis: Pagina de análisis financiero (AnalysisPage).
- /reminders: Pagina de recordatorios (RemindersPage).
- /notifications: Pagina de notificaciones (NotificationsPage).
- /expenses: Pagina de gastos (ExpensesPage).
- /reports: Pagina de reportes (ReportsPage).
- /education: Pagina de educación (EducationPage).
- /user-overview: Resumen del usuario (UserOverviewPage).
- /user-configuration: Configuración de usuario (UserConfigurations).

3. Todas estas rutas están envueltas en NotificationProvider y GlobalNotification para manejar las notificaciones y están protegidas por el componente ProtectedRoute que verifica si el usuario tiene el rol adecuado.

4. Rutas Protegidas para Administradores (Role: "admin"):

- /admin-overview: Vista general del administrador (AdminOverviewPage).
- /user-management: Gestión de usuarios (UserManagementPage).
- /admin-reports: Gestión de reportes del administrador (AdminReportsPage).
- /admin-education: Gestión de la sección de educación para administradores (EducationAdminPage).
- /expense-categories: Gestión de categorías de gastos (ExpenseCategories).
- /goals-management: Gestión de metas para administradores (GoalsManagement).

Comportamiento del Componente

- AuthProvider:
 - Envuelve la aplicación para proporcionar acceso al estado de autenticación y funciones relacionadas con el manejo de la autenticación.
- Router y Rutas:
 - Usa BrowserRouter para el enrutamiento y Routes para definir cada ruta de la aplicación.
 - El componente ProtectedRoute asegura que solo los usuarios autenticados con el rol requerido puedan acceder a ciertas rutas.
- NotificationProvider:
 - Envuelve las rutas de usuario y proporciona un contexto para manejar notificaciones globales, lo que permite que cada página pueda mostrar notificaciones según las acciones del usuario.
- GlobalNotification:
 - Componente que muestra notificaciones globales basadas en el estado de la aplicación.

Proceso de Autenticación y Protección de Rutas

- ProtectedRoute:
 - Verifica el rol del usuario actual y permite o restringe el acceso a una ruta.
 - Si un usuario no está autenticado o no tiene el rol requerido, se redirige a la página de inicio de sesión o se muestra un mensaje de acceso denegado.


```
34  /* Rutas públicas */
35  <Route path="/" element={<HomePage />} />
36  <Route path="/login" element={<login />} />
37  <Route path="/register" element={<Register />} />
38  <Route path="/how-it-works" element={<HowItWorks />} />
39  <Route path="/prices" element={<PricingTier />} />
40  <Route path="/about-us" element={<AboutSection />} />
41
42  /* Rutas de usuario, cada una envuelta en NotificationProvider y GlobalNotification */
43  <Route
44    path="/goals"
45    element={
46      <NotificationProvider>
47        <GlobalNotification />
48        <ProtectedRoute requiredRole="usuario">
49          <GoalsPage />
50        </ProtectedRoute>
51      </NotificationProvider>
52    }
53  />
54  <Route
55    path="/analysis"
56    element={
57      <NotificationProvider>
58        <GlobalNotification />
59        <ProtectedRoute requiredRole="usuario">
```

```

    <ProtectedRoute requiredRole="usuario">
      <GoalsPage />
    </ProtectedRoute>
  </NotificationProvider>
</Route>
<Route
  path="/analysis"
  element={
    <NotificationProvider>
      <GlobalNotification />
      <ProtectedRoute requiredRole="usuario">
        <AnalysisPage />
      </ProtectedRoute>
    </NotificationProvider>
  }
</Route>
<Route
  path="/expenses"
  element={
    <NotificationProvider>
      <GlobalNotification />
      <ProtectedRoute requiredRole="usuario">
        <ExpensesPage />
      </ProtectedRoute>
    </NotificationProvider>
  }
</Route>
```

FrontEnd

- Archive: newsAPI.js
- Este archive contiene la implementación de una función que interactúa con la API de NewsAPI para obtener artículos de noticias basados en una palabra clave.
- Descripción General
 - API Utilizada: NewsAPI, una API que permite acceder a artículos de noticias de diversas fuentes en tiempo real.
 - Clave API: Se utiliza una clave de API específica (API_KEY) para autenticar las solicitudes a la API de NewsAPI.
- Variables y Constantes
 - API_KEY: Clave de API necesaria para la autenticación. Debe reemplazarse con una clave válida al usar en producción.
 - BASE_URL: URL base de la API de NewsAPI para realizar las solicitudes.
- Funciones

```
export async function fetchArticles(keyword = "finance") {  
  try {
```

Descripción: Función asíncrona que obtiene artículos de noticias de la API de NewsAPI basados en una palabra clave proporcionada.

Parámetros:

- keyword (string, opcional): La palabra clave para buscar artículos. Por defecto, es "finance".

Retorno: Devuelve una promesa que resuelve en un array de artículos de noticias si la solicitud es exitosa. En caso de error, devuelve un array vacío.

Uso:

- Realiza una solicitud HTTP GET a la URL base de la API, incluyendo parámetros de búsqueda, como la palabra clave, el idioma (es para español) y el orden de publicación (publishedAt).
- La respuesta se verifica para asegurarse de que sea ok (código de estado 200). Si no lo es, se lanza un error.
- Si la solicitud es exitosa, se extraen los artículos del objeto de respuesta JSON y se devuelven.
- En caso de error, se captura y se imprime en la consola, devolviendo un array vacío.

FrontEnd

- Archivo: CategoryExpensesChart.jsx
- Este archivo define un componente de React que muestra un gráfico de barras para representar los gastos categorizados, utilizando la biblioteca recharts.
- Dependencias
 - PropTypes: Utilizado para definir los tipos de las propiedades recibidas por el componente.
 - recharts: Biblioteca de gráficos en React utilizada para construir el grafico de barras.
 - BarChart, Bar, XAxis, YAxis, Tooltip, ResponsiveContainer, Cell, Legend: Componentes importados de recharts para la creaci6n y personalizaci6n del gráfico.
- Descripción del Componente
 - Nombre del Componente: CategoryExpensesChart
 - Props:
 - data (array, requerido): Array de objetos que representan las categorfas y los montos de los gastos. Cada objeto debe tener las propiedades:
 - name (string): Nombre de la categoría.
 - amount (number): Monto del gasto asociado a la categoría.
- Funcionalidades y Caracterfsticas
 - a. Definici6n de Gradientes:
 - Se define un array de colores de gradiente (gradientColors) que se utiliza para asignar un color espedfico a cada barra del gráfico.
 - b. Leyenda Personalizada:
 - renderLegend es una funci6n que renderiza una leyenda personalizada centrada. Muestra los nombres de las categorfas con un icono de color que representa el gradiente correspondiente.
 - c. Contenedor de Grafico:
 - Se utiliza ResponsiveContainer para garantizar que el grafico sea responsivo y se ajuste al contenedor padre.
 - d. Configuraci6n del Grafico de Barras:
 - defs: Se definen los gradientes de colores utilizando elementos <linearGradient>.
 - XAxis y YAxis: Configuran los ejes Xe Y del gráfico.
 - Tooltip: Personaliza la visualizaci6n de la informaci6n emergente cuando el usuario pasa el cursor sobre las barras.
 - Legend: Se pasa renderLegend como contenido de la leyenda para usar la leyenda personalizada.
 - Bar:
 - Representa las barras del gráfico, donde cada barra tiene un relleno de color determinado por un gradiente definido en defs.
 - Cell: Utilizado para asignar a cada barra un color espedfico del gradiente.
- Validaci6n de Propiedades
 - Se utiliza PropTypes para asegurar que data sea un array de objetos con las propiedades name (string) y amount (numero).
- Descripción Visual
 - Gráfico de Barras: Cada barra representa una categoría de gasto con un color de gradiente único. Los colores est6n definidos en gradientColors y se aplican de manera única si hay m6s barras que gradientes.
 - Leyenda Personalizada: Muestra una representaci6n visual de los colores de las barras junto con el nombre de la categorfa, centrada debajo del gráfico.

```
gradientColors = [  
  { "gradient1", start: "#8B5CF6", end: "#6D28D9" },  
  { "gradient2", start: "#3B82F6", end: "#2266BB" },  
  { "gradient3", start: "#22C55E", end: "#10B981" },  
  { "gradient4", start: "#F44336", end: "#E57373" },  
]  
  
// Leyenda personalizada centrada con representaci6n de gradientes  
renderLegend = () => (  
  <div style={{ display: "flex", justify-content: "center", align-items: "center", margin: 10 }}>  
    <div style={{ display: "flex", listStyleType: "none", padding: 0, gap: "15px" }}>  
      {data.map((entry, index) => (  
        <li key={`item-${index}`} style={{ display: "flex", align-items: "center" }}>  
          <div style={{ width: "14" height: "14" }}>  
            <defs>  
              <linearGradient id={`legend-gradient-${index}`} x1="0" y1="0" x2="1" y2="1">  
                <stop offset="0%" stopColor={gradientColors[index % gradientColors.length].start} stopOpacity={0}>  
                <stop offset="100%" stopColor={gradientColors[index % gradientColors.length].end} stopOpacity={1}>  
              </linearGradient>  
            </defs>  
            <rect width="14" height="14" fill={`url(#legend-gradient-${index})`} />  
          </div>  
          <span style={{ margin: 0 5 }}>{entry.name}</span>  
        </li>  
      )>  
    </div>  
  )>  
)
```

FrontEnd

- Archive: ExpensesDistributionChart.jsx
- Este archive define un componente de React que muestra un gráfico de pastel para visualizar la distribución de los gastos, utilizando la biblioteca recharts.
- Dependencias
 - PropTypes: Utilizado para definir los tipos de las propiedades recibidas por el componente.
 - recharts: Biblioteca de gráficos en React que se utiliza para construir gráficos interactivos.
 - PieChart, Pie, Cell, Tooltip, ResponsiveContainer: Componentes importados de recharts para construir y personalizar el gráfico de pastel.
- Descripción del Componente
 - Nombre del Componente: ExpensesDistributionChart
 - Props:
 - data (array, requerido): Array de objetos que representan la distribución de los gastos. Cada objeto debe tener las propiedades:
 - name (string): Nombre de la categoría de gasto.
 - value (number): Valor numérico del gasto en esa categoría.
- Funcionalidades y Características
 - a. Definición de Gradientes:
 - Se definen gradientes de colores (<linearGradient>) para dar un aspecto visual atractivo a las secciones del gráfico de pastel. Cada gradiente se aplica a una sección específica del gráfico, y se usan de forma cíclica si hay más secciones que gradientes.
 - b. Contenedor de Gráfico:
 - ResponsiveContainer: Permite que el gráfico se ajuste automáticamente al tamaño del contenedor padre, garantizando que el gráfico sea responsive.
 - c. Configuración del Gráfico de Pastel:
 - Pie:
 - Muestra los datos como un gráfico de pastel.
 - dataKey: Clave de los datos utilizada para el valor de cada sector del pastel.
 - nameKey: Clave de los datos utilizada para el nombre de cada sector.
 - ex y cy: Posicionan el centro del gráfico en el contenedor.
 - outerRadius: Define el radio externo del gráfico.
 - Cell:
 - Se utiliza para aplicar los gradientes de colores a cada sector del gráfico de pastel.
 - Asigna un color de gradiente a cada celda utilizando la función fill.
 - d. Tooltip:
 - Se incluye un componente Tooltip que muestra información sobre la sección seleccionada cuando el usuario pasa el cursor sobre ella.
- Validación de Propiedades
 - PropTypes:
 - Asegura que data sea un array de objetos con las propiedades name (string) y value (numero).
- Descripción Visual
 - Gráfico de Pastel: Cada sección representa una categoría de gasto con un color de gradiente distinto, lo que hace que el gráfico sea fácil de interpretar y visualmente atractivo.
 - Gradientes: Los colores de gradiente proporcionan una apariencia moderna y diferenciada para cada sección del gráfico.

```
/* Definición de gradientes para cada sección */
<defs>
  <linearGradient id="gradient1" x1="0" y1="0" x2="1" y2="1">
    <stop offset="0%" stopColor="#8b5cf6" stopOpacity={0.8} />
    <stop offset="100%" stopColor="#6d28d9" stopOpacity={1} />
  </linearGradient>
  <linearGradient id="gradient2" x1="0" y1="0" x2="1" y2="1">
    <stop offset="0%" stopColor="#3b82f6" stopOpacity={0.8} />
    <stop offset="100%" stopColor="#2563eb" stopOpacity={1} />
  </linearGradient>
  <linearGradient id="gradient3" x1="0" y1="0" x2="1" y2="1">
    <stop offset="0%" stopColor="#22c55e" stopOpacity={0.8} />
    <stop offset="100%" stopColor="#16a34a" stopOpacity={1} />
  </linearGradient>
  <linearGradient id="gradient4" x1="0" y1="0" x2="1" y2="1">
    <stop offset="0%" stopColor="#ef4444" stopOpacity={0.8} />
    <stop offset="100%" stopColor="#b91c1c" stopOpacity={1} />
  </linearGradient>
</defs>
```

- Archivo: GoalProgressChart.jsx
- Este archivo define un componente de React que visualiza el progreso de las metas en un gráfico de pastel, utilizando la biblioteca recharts.
- Dependencias
 - PropTypes: Utilizado para la validación de las propiedades recibidas por el componente.
 - recharts: Biblioteca de gráficos para React.
 - PieChart, Pie, Cell, ResponsiveContainer, Tooltip: Componentes utilizados para construir el grafico de pastel.
- Descripción del Componente
 - Nombre del Componente: GoalProgressChart
 - Props:
 - goals (array, requerido): Array de objetos que representan las metas. Cada objeto debe tener las propiedades:
 - name (string): Nombre de la meta.
 - targetAmount (number): Monto objetivo de la meta.
 - amountAchieved (number): Monto alcanzado de la meta.
- Funcionalidades y Características
 - a. Verificación de Metas Disponibles:
 - Si no hay metas o la lista esta vacía, el componente devuelve un mensaje indicando que no hay metas disponibles.
 - b. Cálculo de Progreso:
 - Se calcula el porcentaje de progreso dividiendo el amountAchieved por el targetAmount y multiplicando por 100. Se asegura que el porcentaje máxima sea 100% para evitar desbordamientos visuales.
 - c. Datos del Grafico:
 - El componente define los datos para el grafico de pastel en dos secciones:
 - "Logrado": Representa el porcentaje de la meta alcanzada.
 - "Restante": Representa el porcentaje restante hasta alcanzar la meta.
 - d. Navegación entre Metas:
 - Se incluyen botones de navegación (prevGoal y nextGoal) para moverse entre las diferentes metas.
 - La navegación es cíclica, es decir, al llegar al final de la lista, se regresa al inicio y viceversa.
 - e. Configuración del Grafico de Pastel:
 - Pie:
 - Representa las secciones del gráfico, con una sección para el progreso y otra para el restante.
 - Utiliza un degradado de color verde para la parte de "Logrado" y un color gris para "Restante".
 - Cell:
 - Asigna el color degradado y el color de fondo para cada sección del gráfico.
 - Tooltip:
 - Muestra un tooltip con el porcentaje formateado cuando el usuario pasa el cursor sobre el grafico.
 - f. Leyenda y Gradientes:
 - Se define un gradiente (<linearGradient>) para el color de la celda de progreso, proporcionando una visualización más atractiva.

FrontEnd

- Validaci6n de Propiedades
 - PropTypes:
 - Asegura que goals sea un array de objetos con las propiedades name, targetAmount, y amountAchieved, todos requeridos.
- Descripci6n Visual
 - Gráfico de Pastel:
 - El grafico muestra dos secciones: el progreso logrado y el restante hasta la meta.
 - Utiliza un degradado verde para la parte lograda y un color gris para la parte restante.
 - Navegaci6n de Metas:
 - Botones para navegar entre las metas, permitiendo al usuario visualizar diferentes metas y sus respectivos porcentajes de progreso.
 - Tooltip:
 - Proporciona informaci6n detallada sobre el porcentaje cuando el usuario interactúa con el grafico.

```
const [currentGoalIndex, setCurrentGoalIndex] = useState(0);

// Verificar si hay metas disponibles
if (!goals || goals.length === 0) {
  return <p>No hay metas disponibles.</p>;
}

const currentGoal = goals[currentGoalIndex];
const percentage = Math.min(
  (currentGoal.amountAchieved / currentGoal.targetAmount) * 100,
  100
);

const data = [
  { name: "Logrado", value: percentage },
  { name: "Restante", value: 100 - percentage },
];

const nextGoal = () => {
  setCurrentGoalIndex((prevIndex) => (prevIndex + 1) % goals.length);
};

const prevGoal = () => {
  setCurrentGoalIndex(
    (prevIndex) => (prevIndex - 1 + goals.length) % goals.length
  );
};
```

```
<div style={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}>
  <button onClick={prevGoal}><img alt="Previous Goal Icon" data-bbox="330 610 345 625"/></button>
  <ResponsiveContainer width={250} height={250}>
    <PieChart>
      <defs>
        <linearGradient id="goalProgressGradient" x1="0" y1="0" x2="1" y2="1">
          <stop offset="0%" stopColor="#21AA58" stopOpacity={1} />
          <stop offset="63%" stopColor="#21AA58" stopOpacity={1} />
          <stop offset="100%" stopColor="#247D61" stopOpacity={1} />
        </linearGradient>
      </defs>
      <Pie>
        data={data}
        dataKey="value"
        innerRadius={70}
        outerRadius={90}
        startAngle={90}
        endAngle={-270}
      >
      <Cell fill="url(#goalProgressGradient)" /> { /* Color verde degradado */}
      <Cell fill="#e5e7eb" /> { /* Color gris para el restante */}
    </Pie>
    <Tooltip formatter={(value) => `${goals.targetAmount} - ${value}%`} />
  </ResponsiveContainer>
  <button onClick={nextGoal}><img alt="Next Goal Icon" data-bbox="575 610 590 625"/></button>
</div>
```

- Archivo: IncomeExpensesPieChart.jsx
- Este archivo define un componente de React que muestra un gráfico de pastel para comparar los ingresos y los gastos, utilizando la biblioteca recharts.
- Dependencias
 - PropTypes: Utilizado para la validación de las propiedades recibidas por el componente.
 - recharts: Biblioteca de gráficos para React.
 - PieChart, Pie, Cell, Tooltip, ResponsiveContainer, Legend: Componentes utilizados para construir y personalizar el gráfico de pastel.
- Descripción del Componente
 - Nombre del Componente: IncomeExpensesPieChart
 - Props:
 - income (number, requerido): Valor numérico que representa los ingresos totales.
 - totalExpenses (number, requerido): Valor numérico que representa los gastos totales.
- Funcionalidades y Características
 - a. Preparación de los Datos:
 - Se crea un array de objetos data que contiene los datos de ingresos y gastos, cada uno con un nombre, un valor y un color que se asigna mediante un gradiente.
 - b. Definición de Gradientes:
 - Se utilizan elementos `<linearGradient>` para definir los gradientes de color de las secciones de ingresos y gastos, proporcionando un estilo visual atractivo.
 - Cada gradiente se asigna un identificador (id) único para ser referenciado en los elementos del gráfico.
 - c. Configuración del Gráfico de Pastel:
 - Pie:
 - Representa las secciones del gráfico de pastel con los datos de ingresos y gastos.
 - ex y cy: Posicionan el gráfico en el centro del contenedor.
 - outerRadius y innerRadius: Definen el radio externo e interno del gráfico, creando un efecto de dona.
 - dataKey: Define la clave de los datos utilizada para representar los valores en el gráfico.
 - Cell:
 - Aplica los colores de gradiente a cada sección del gráfico de pastel mediante la referencia a los identificadores definidos en `<linearGradient>`.
 - d. Elementos Adicionales:
 - Tooltip: Muestra un tooltip interactivo con el valor formateado cuando el usuario pasa el cursor sobre el gráfico.
 - Legend: Muestra una leyenda que identifica cada sección del gráfico (ingresos y gastos).
 - e. Contenedor Responsivo:
 - ResponsiveContainer: Garantiza que el gráfico se ajuste al tamaño del contenedor padre, manteniendo su responsividad.
- Validación de Propiedades
 - PropTypes:
 - Asegura que income y totalExpenses sean valores numéricos requeridos.
- Descripción Visual
 - Gráfico de Pastel:
 - El gráfico de dona muestra dos secciones: una para los ingresos y otra para los gastos, con colores distintivos definidos por gradientes.
 - Gradientes de Color:
 - Los gradientes proporcionan una representación visual atractiva, diferenciando claramente las secciones de ingresos y gastos.
 - Tooltip y Leyenda:
 - Proporcionan información interactiva al usuario y una representación clara de que parte del gráfico corresponde a cada tipo de dato.

- Archivo: AddAmountModal.jsx
- Este archivo define un componente de modal en React que permite al usuario agregar un monto adicional a una meta financiera.
- Dependencias
 - useState: Hook de React utilizado para manejar el estado del monto adicional ingresado por el usuario.
 - PropTypes: Utilizado para la validaci6n de las propiedades recibidas por el componente.
- Descripci6n del Componente
 - Nombre del Componente: AddAmountModal
 - Props:
 - isOpen (bool, requerido): Indica si el modal debe estar abierto o cerrado.
 - onClose (func, requerido): Funci6n para cerrar el modal.
 - onSave (func, requerido): Funci6n que se ejecuta al guardar el monto adicional ingresado.
 - montoActual (number, requerido): Valor actual de la meta, mostrado en el modal como referencia.
- Funcionalidades y Características
 - a. Estado interno:
 - montoAdicional: Maneja el valor del monto adicional que el usuario ingresa en el campo de texto.
 - b. Manejo de Eventos:
 - handleSave: Funci6n que se ejecuta al enviar el formulario, llama a onSave con el monto adicional ingresado y cierra el modal con onClose.
 - c. Renderizaci6n Condicional:
 - Si isOpen es false, el componente retorna null y no se renderiza.
 - d. interfaz de Usuario:
 - Un modal centrado con un fondo semitransparente que contiene un formulario con:
 - Un t6tulo y una descripci6n que muestra el monto actual.
 - Un campo de entrada num6rica para que el usuario ingrese el monto adicional.
 - Botones de "Cancelar" y "Guardar" para manejar la acci6n del usuario.
 - El bot6n "Guardar" env6a el formulario y ejecuta la funci6n onSave.
- Validaci6n de Propiedades
 - PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una funci6n y es obligatoria.
 - onSave: Debe ser una funci6n y es obligatoria.
 - montoActual: Debe ser un n6mero y es obligatorio.
- Descripci6n Visual
 - Modal:
 - Se renderiza como una ventana emergente centrada en la pantalla con un fondo de superposici6n gris semitransparente.
 - Formulario:
 - Contiene un campo de entrada de tipo num6rico con validaci6n para que no se ingresen valores negativos.
 - Botones:
 - Bot6n "Cancelar" para cerrar el modal sin guardar cambios.
 - Bot6n "Guardar" para enviar el formulario y llamar a la funci6n onSave.

Nombre

Descripci6n

Fecha Objetivo

dd/mm/aaaa

Monto Objetivo

Categor6a

Seleccionar categor6a

- Archivo: AddExpenseModal.jsx
- Este archivo define un componente de modal en React para agregar un nuevo gasto a la aplicación.
- Dependencias
 - useState, useEffect, useContext: Hooks de React para manejar el estado, efectos secundarios y contexto.
 - AuthContext: Contexto para acceder al token de autenticación.
 - agregarGasto: Función de la API para agregar un gasto.
 - obtenerCategoriasGasto: Función de la API para obtener las categorías de gastos.
 - getIconForCategory: Función de utilidad para obtener iconos de categorías.
 - Select: Componente de selección de la biblioteca react-select.
 - PropTypes: Validación de las propiedades del componente.
- Descripción del Componente
 - Nombre del Componente: AddExpenseModal
 - Props:
 - isOpen (bool, requerido): Indica si el modal está visible.
 - onClose (func, requerido): Función para cerrar el modal.
 - onExpenseAdded (func, requerido): Función que se ejecuta cuando se agrega un gasto exitosamente.
- Funcionalidades y Características
 - a. Estado interno:
 - categories: Almacena las categorías de gastos obtenidas de la API.
 - monto: Monto del gasto ingresado por el usuario.
 - selectedCategory: Categoría seleccionada por el usuario.
 - descripcion: Descripción del gasto.
 - b. Manejo de Efectos:
 - useEffect para cargar las categorías de gastos cuando el componente se monta.
 - c. Funciones:
 - fetchCategories: Llama a obtenerCategoriasGasto para obtener las categorías y las guarda en el estado categories.
 - handleAddExpense: Envía los datos del gasto a la API usando agregarGasto, llama a onExpenseAdded y cierra el modal.
 - parseJwt: Decodifica el token JWT para obtener el ID del usuario.
 - d. Renderización Condicional:
 - Retorna null si isOpen es false para no mostrar el modal.
 - e. Interfaz de Usuario:
 - Un modal con un formulario que incluye:
 - Campo de entrada numérica para el monto.
 - Un componente Select para elegir la categoría, con iconos personalizados para cada opción.
 - Un campo de texto para la descripción del gasto.
 - Botones de "Cancelar" y "Agregar Gasto" para las acciones del usuario.
- Validación de Propiedades
 - PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - onExpenseAdded: Debe ser una función y es obligatoria.
- Descripción Visual
 - **Modal:**
 - Se presenta como una ventana emergente centrada con un fondo de superposición.
 - Formulario:
 - Contiene campos con estilos y validaciones para garantizar la entrada de datos adecuada.
 - Botones:
 - "Cancelar" para cerrar el modal sin guardar los cambios.
 - "Agregar Gasto" para enviar el formulario y llamar a la función onExpenseAdded.

Archivo: AddGoalModal.jsx

Este archivo define un componente de modal en React para agregar una nueva meta financiera a la aplicaci6n.

Dependencias

- useState, useEffect, useContext: Hooks de React para manejar el estado, efectos secundarios y contexto.
- AuthContext: Contexto para acceder al token de autenticaci6n del usuario.
- agregarMeta: Funci6n de la API para agregar una meta.
- obtenerCategoriasMeta: Funci6n de la API para obtener las categorfas de metas.
- getIconForCategory: Funci6n de utilidad para obtener iconos asociados a categorfas.
- Select: Componente de la biblioteca react-select para listas desplegables.
- PropTypes: Validaci6n de las propiedades del componente.

Descripci6n del Componente

- Nombre del Componente: AddGoalModal
- Props:
 - isOpen (bool, requerido): Indica si el modal debe mostrarse.
 - onClose (func, requerido): Funci6n para cerrar el modal.
 - onGoalAdded (func, requerido): Funci6n que se ejecuta cuando se agrega una meta exitosamente.

Funcionalidades y Caracterfsticas

7. Estado interno:

- nombre, descripcion, fechaObjetivo, montoObjetivo: Controlan las campos del formulario.
- categories: Almacena las categorfas de metas obtenidas de la API.
- selectedCategory: Almacena la categorfa seleccionada para el usuario.

8. Manejo de Efectos:

- useEffect para cargar las categorfas desde el backend cuando el componente se monta.

9. Funciones:

- fetchCategories: Llama a obtenerCategoriasMeta para obtener las categorfas de metas y las guarda en el estado categories.
- handleAddGoal: Entra las datos de la meta a la API usando agregarMeta, llama a onGoalAdded y cierra el modal.
- parseJwt: Decodifica el token JWT para obtener el ID del usuario autenticado.

10. Renderizaci6n Condicional:

- Retorna null si isOpen es false para no mostrar el modal.

11. interfaz de Usuario:

- Un modal con un formulario que incluye:
 - Campos de entrada para el nombre, descripci6n, fecha objetivo y monto objetivo de la meta.
 - Un componente Select para elegir la categorfa de la meta, mostrando iconos personalizados.
 - Botones de "Cancelar" y "Agregar Meta" para las acciones del usuario.

Validaci6n de Propiedades

- PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una funci6n y es obligatoria.
 - onGoalAdded: Debe ser una funci6n y es

obligatoria. Descripci6n Visual

- **Modal:**
 - Presenta un fondo de superposici6n semitransparente y un formulario centrado.
- Formulario:
 - Dise1ado con campos de entrada estilizados y validaciones para garantizar la correcta entrada de datos.
- Botones:
 - "Cancelar" cierra el modal sin guardar los cambios.
 - "Agregar Meta" entra el formulario y ejecuta la funci6n onGoalAdded para actualizar la lista de metas.

FrontEnd

Archivo: AddReminderModal.jsx

Este archivo define un componente de modal en React que permite a los usuarios agregar un nuevo recordatorio.

Dependencias

- `useState` y `useContext`: Hooks de React para manejar el estado y el contexto.
- `AuthContext`: Contexto de autenticación para obtener el token del usuario.
- `agregarRecordatorio`: Función de la API que permite agregar un recordatorio.
- `PropTypes`: Validación de las propiedades del componente.

Descripción del Componente

- **Nombre del Componente:** `AddReminderModal`
- **Props:**
 - `isOpen` (bool, requerido): Indica si el modal está visible.
 - `onClose` (func, requerido): Función para cerrar el modal.
 - `onReminderAdded` (func, requerido): Función que se ejecuta al agregar un nuevo recordatorio.

Funcionalidades y Características

7. Estado interno:

- `descripcion`: Estado para almacenar la descripción del recordatorio.
- `fechaRecordatorio`: Estado para almacenar la fecha del recordatorio.

8. Funciones:

- `handleAddReminder`:
 - Se ejecuta al enviar el formulario.
 - Llama a `agregarRecordatorio`, pasando el token y los datos del recordatorio.
 - Llama a `onReminderAdded` para informar al componente padre.
 - Cierra el modal después de guardar.
- `parseJwt`:
 - Decodifica el token JWT para obtener el `usuario_id`.

9. Configuración de Fecha:

- La fecha mínima en el campo de entrada de fecha se establece en la fecha actual para evitar la selección de fechas pasadas.

10. Renderización Condicional:

- Retorna null si `isOpen` es false, evitando que se muestre el modal.

11. Interfaz de Usuario:

- Un modal con:
 - Un campo de entrada de texto para la descripción del recordatorio.
 - Un campo de entrada de fecha para seleccionar la fecha del recordatorio.
 - Botones de "Cancelar" y "Agregar Recordatorio" para las acciones del usuario.

Validación de Propiedades

- `PropTypes`:
 - `isOpen`: Debe ser un booleano y es obligatorio.
 - `onClose`: Debe ser una función y es obligatoria.
 - `onReminderAdded`: Debe ser una función y es obligatoria.

Descripción Visual

- **Modal:**
 - Se muestra centrado en la pantalla con un fondo semitransparente que cubre toda la ventana.
- **Formulario:**
 - Contiene un área de texto para que el usuario ingrese la descripción.
 - Un campo de entrada de fecha que solo permite fechas futuras o actuales.
- **Botones:**
 - "Cancelar" cierra el modal sin guardar.
 - "Agregar Recordatorio" guarda el nuevo recordatorio y cierra el modal.



The screenshot shows a modal titled "Agregar Recordatorio". It contains a text input field labeled "Descripción", a date input field labeled "Fecha del Recordatorio" with a placeholder "dd/mm/aaaa" and a calendar icon, and two buttons at the bottom: "Cancelar" and "Agregar Recordatorio".

FrontEnd

Archivo: AddReportModal.jsx

Este archivo contiene un componente de modal en React que permite a los usuarios añadir nuevos reportes.

Dependencias

- useState y useContext: Hooks de React para manejar el estado y el contexto.
- PropTypes: Validación de las propiedades del componente.
- AuthContext: Contexto de autenticación para obtener el token del usuario.
- crearReporte: Función de la API que permite crear un nuevo reporte.

Descripción del Componente

- Nombre del Componente: AddReportModal
- Props:
 - isOpen (bool, requerido): Indica si el modal esta visible.
 - onClose (func, requerido): Función para cerrar el modal.
 - onReportAdded (func, requerido): Función que se ejecuta cuando se añade un nuevo reporte.

Funcionalidades y Características

7. Estado interno:

- título: Estado para almacenar el título del reporte.
- descripcion: Estado para almacenar la descripción del reporte.

8. Funciones:

- handleSave:
 - Se ejecuta al enviar el formulario.
 - Obtiene el usuario_id del token mediante parseJwt.
 - Llama a crearReporte, pasando el token y los datos del reporte.
 - Llama a onReportAdded para informar al componente padre de la adición del reporte.
 - Cierra el modal después de guardar.
- parseJwt:
 - Decodifica el token JWT para extraer el usuario_id.

9. Renderización Condicional:

- Si isOpen es false, el modal no se renderiza.

10. interfaz de Usuario:

- Modal con:
 - Campo de entrada de texto para el título del reporte.
 - Área de texto para la descripción del reporte.
 - Botones "Cancelar" y "Guardar" para manejar las acciones del usuario.

Validación de Propiedades

- PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - onReportAdded: Debe ser una función y es obligatoria.

Descripción Visual

- Modal:
 - Se muestra centrado en la pantalla con un fondo semitransparente que cubre toda la ventana.
- Formulario:
 - Contiene campos de entrada para el título y la descripción del reporte.
- Botones:
 - "Cancelar": Cierra el modal sin guardar.
 - "Guardar": Guarda el reporte y cierra el modal.

Añadir Reporte

Título

Descripción

Cancelar Guardar

FrontEnd

Archivo: EditExpenseModal.jsx

Este archivo contiene un componente de React que proporciona una interfaz para editar un gasto existente. El modal muestra un formulario que permite al usuario actualizar detalles como el monto, la categoría y la descripción de un gasto.

Dependencias

- `useState`, `useEffect`, `useContext`: Hooks de React para manejar el estado, efectos y contexto.
- `PropTypes`: Validación de tipos de las propiedades del componente.
- `AuthContext`: Contexto de autenticación para obtener el token del usuario.
- `obtenerCategoriasGasto`, `editarGasto`: Funciones de la API para obtener categorías de gasto y editar un gasto.
- `getIconForCategory`: Función utilitaria para obtener el icono correspondiente a una categoría.
- `Select`: Componente de selección de `react-select` para mostrar y seleccionar categorías.

Descripción del Componente

- Nombre del Componente: `EditExpenseModal`
- Props:
 - `isOpen` (bool, requerido): Determina si el modal se muestra.
 - `onClose` (func, requerido): Función que cierra el modal.
 - `gasto` (object, requerido): Objeto que contiene los datos del gasto a editar.
 - `onExpenseUpdated` (func, requerido): Función que se llama después de que el gasto ha sido editado exitosamente.

Funcionalidades y Características

7. Estado Interno:

- `categories`: Lista de categorías de gasto obtenidas desde la API.
- `monto`, `selectedCategory`, `descripcion`: Valores del gasto a editar, inicializados con los datos del prop `gasto`.

8. Funciones:

- `useEffect`:
 - Carga las categorías de gasto al montar el componente.
- `handleEditExpense`:
 - Envía los datos actualizados del gasto a la API y cierra el modal al completar la operación.
- `parseJwt`:
 - Decodifica el token JWT para obtener el `usuario_id`.

9. Renderización Condicional:

- Si `isOpen` es `false`, el modal no se muestra.

10. Interfaz de Usuario:

- Formulario que incluye:
 - Campo de entrada para el monto.
 - Selección de categoría con iconos.
 - Área de texto para la descripción.
- Botones de acción: "Cancelar" y "Guardar Cambios".

Validación de Propiedades

- `PropTypes`:
 - `isOpen`: Debe ser un booleano y es obligatorio.
 - `onClose`: Debe ser una función y es obligatoria.
 - `gasto`: Debe ser un objeto que representa el gasto a editar y es obligatorio.
 - `onExpenseUpdated`: Debe ser una función y es obligatoria.

Descripción Visual

- Modal:
 - Se muestra centrado en la pantalla con un fondo semitransparente.
 - Contenedor blanco con bordes redondeados y sombra.
- Formulario:
 - Incluye campos editables para el monto, la categoría y la descripción.
- Botones:
 - "Cancelar": Permite cerrar el modal sin guardar cambios.
 - "Guardar Cambios": Envía el formulario y guarda los cambios en el gasto.

archivo: EditGoalModal.jsx

Este componente es un modal de React que permite al usuario editar una meta existente. Proporciona un formulario con campos pre-rellenados con la información de la meta que se quiere actualizar y permite guardar los cambios.

Dependencias

- useState: Hook de React para manejar el estado local.
- PropTypes: Librería para la validación de las propiedades del componente.

Descripción del Componente

- Nombre del Componente: EditGoalModal
- Props:
 - isOpen (bool, requerido): Indica si el modal debe mostrarse o no.
 - onClose (func, requerido): Función que cierra el modal.
 - onSave (func, requerido): Función que se ejecuta al guardar los cambios, pasando los datos actualizados de la meta.
 - meta (object, requerido): Objeto que contiene la información de la meta a editar, incluyendo:
 - nombre_meta: Nombre de la meta.
 - monto_objetivo: Monto objetivo de la meta.
 - monto_actual: Monto actual alcanzado de la meta.
 - descripcion: Descripción de la meta.
 - estado_de_meta: Estado actual de la meta (por ejemplo, "activo", "completado", "cancelado").

Funcionalidades y Características

7. Estado interno:
 - nombreMeta, montoObjetivo, montoActual, descripcion, estadoDeMeta: Variables de estado que almacenan los valores de los campos editables.
8. Funciones:
 - handleSave:
 - Maneja la lógica de guardado de la meta editada, llama a onSave con los datos actualizados y cierra el modal.
9. Renderización Condicional:
 - Si isOpen es false, el componente retorna null y no se muestra en pantalla.
10. Interfaz de Usuario:
 - Formulario que incluye:
 - Campo de entrada para el nombre de la meta.
 - Campo de entrada numérico para el monto objetivo.
 - Campo de entrada numérico para el monto actual.
 - Área de texto para la descripción.
 - Menú desplegable para seleccionar el estado de la meta.
 - Botones de acción: "Cancelar" y "Guardar Cambios".

Validación de Propiedades

- PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - onSave: Debe ser una función y es obligatoria.
 - meta: Debe ser un objeto con las siguientes propiedades:
 - nombre_meta: Cadena de texto, obligatoria.
 - monto_objetivo: Número, obligatorio.
 - monto_actual: Número, obligatorio.
 - descripcion: Cadena de texto, obligatoria.
 - estado_de_meta: Cadena de texto, obligatoria.

Descripción Visual

- Modal:
 - Se muestra centrado en la pantalla con un fondo semitransparente.
 - Contenedor blanco con bordes redondeados y sombra.
- Formulario:
 - Campos editables con estilos básicos para ingresar el nombre, montos, descripción y estado de la meta.
- Botones:
 - "Cancelar": Permite cerrar el modal sin guardar los cambios.
 - "Guardar Cambios": Envía el formulario y actualiza los datos de la meta.

Archivo: EditReminderModal.jsx

Este componente es un modal de React que permite al usuario editar un recordatorio existente. Proporciona un formulario que incluye las detalles pre-rellenados del recordatorio seleccionado y permite al usuario actualizar la información.

Dependencias

- useState: Hook de React para manejar el estado local.
- useEffect: Hook de React para manejar efectos secundarios.
- useContext: Hook de React para usar el contexto global de la aplicación.
- AuthContext: Contexto de autenticación que provee el token de usuario.
- editarRecordatorio: Función de la API para actualizar un recordatorio existente.
- PropTypes: Librería para la validación de las propiedades del componente.

Descripción del Componente

- Nombre del Componente: EditReminderModal
- Props:
 - isOpen (bool, requerido): Indica si el modal debe mostrarse o no.
 - onClose (función, requerido): Función que cierra el modal.
 - recordatorio (object, opcional): Objeto que contiene la información del recordatorio a editar.
 - recordatorio_id: ID del recordatorio.
 - descripcion: Descripción del recordatorio.
 - fecha_recordatorio: Fecha programada del recordatorio.
 - onReminderUpdated (función, requerido): Función que se ejecuta después de que se edita el recordatorio.

Funcionalidades y Características

7. Estado interno:

- descripcion, fechaRecordatorio: Variables de estado que almacenan los valores de los campos editables.

8. Hooks:

- useEffect: Actualiza el estado interno cuando se selecciona un nuevo recordatorio para edición.

9. Funciones:

- parseJwt: Decodifica el token para extraer el usuario_id.
- handleEditReminder: Maneja el envío del formulario para editar el recordatorio. Llama a editarRecordatorio y, si es exitoso, ejecuta onReminderUpdated y cierra el modal.

10. Renderización Condicional:

- Si isOpen es false, el componente retorna null y no se muestra en la pantalla.

11. Interfaz de Usuario:

- Formulario que incluye:
 - Campo de texto para la descripción del recordatorio.
 - Selector de fecha para la fecha del recordatorio.
- Botones de acción: "Cancelar" y "Guardar Cambios".

Validación de Propiedades

- PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - recordatorio: Debe ser un objeto con las siguientes propiedades opcionales:
 - recordatorio_id: Número.
 - descripcion: Cadena de texto.
 - fecha_recordatorio: Cadena de texto.
 - onReminderUpdated: Debe ser una función y es obligatoria.

Descripción Visual

- Modal:
 - Aparece centrado en la pantalla con un fondo semitransparente.
 - Contenedor blanco con bordes redondeados y sombra.
- Formulario:
 - Incluye campos editables con estilos básicos para ingresar la descripción y la fecha del recordatorio.
- Botones:
 - "Cancelar": Permite cerrar el modal sin guardar los cambios.
 - "Guardar Cambios": Envía el formulario y actualiza los datos del recordatorio.

- Archive: EditModal.jsx
- Este componente es un modal de edición que permite a los usuarios modificar y confirmar la actualización de ciertos datos (correo, ingresos o contraseña). Una vez que el usuario ingresa y confirma los nuevos datos, el componente llama a una función para actualizar la información en la base de datos.
- Dependencias
 - useState: Hook de React para manejar el estado local.
 - motion (de framer-motion): Para animaciones en el modal.
 - PropTypes: Para la validación de las propiedades del componente.
 - actualizarUsuario: Función de la API que se utiliza para actualizar los datos de un usuario.
- Descripción del Componente
 - Nombre del Componente: EditModal
 - Props:
 - isOpen (bool, requerido): Indica si el modal está abierto.
 - onClose (func, requerido): Función que cierra el modal.
 - section (string, requerido): La sección o campo que se está editando (ej., "Correo", "Ingresos", "Contraseña").
 - userId (number, requerido): ID del usuario que se va a actualizar.
 - onSaveSuccess (func, requerido): Función que se llama al guardar exitosamente los cambios.
- Funcionalidades y Características
 - a. Estado interno:
 - value y confirmValue: Almacenan el nuevo valor ingresado por el usuario y su confirmación.
 - error: Almacena mensajes de error que pueden ocurrir durante la validación o la actualización.
 - b. Funciones:
 - handleSave: Verifica si el valor ingresado y su confirmación coinciden. Si es así, llama a la función actualizarUsuario para actualizar los datos en la base de datos. Si la actualización es exitosa, llama a onSaveSuccess y cierra el modal. En caso de error, muestra un mensaje de error.
 - campoMapeado: Mapea la sección proporcionada a su respectivo campo en la base de datos (email, ingresos, o password_usuario).
 - c. Renderización Condicional:
 - Si isOpen es false, el componente retorna null y no se muestra en la pantalla.
 - d. Interfaz de Usuario:
 - Contenedor de fondo semitransparente para resaltar el modal.
 - Campos de entrada para ingresar y confirmar el nuevo valor.
 - Botones de acción:
 - "Cancelar" para cerrar el modal sin realizar cambios.
 - "Guardar" para enviar los nuevos datos y guardar los cambios.
 - e. Validación:
 - Verifica que los valores ingresados y confirmados coincidan antes de permitir la actualización.
 - Si no coinciden, muestra un mensaje de error.
- Validación de Propiedades
 - PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - section: Debe ser una cadena de texto y es obligatoria.
 - userId: Debe ser un número y es obligatorio.
 - onSaveSuccess: Debe ser una función y es obligatoria.

- Descripción Visual
 - Modal:
 - Aparece centrado en la pantalla con un fondo oscuro semitransparente.
 - Contenedor blanco con bordes redondeados y una sombra que mejora la visibilidad.
 - Formulario:
 - Campos estilizados con bordes redondeados y un foco visual al interactuar.
 - Botones:
 - "Cancelar" tiene un fondo gris claro y cambia a un tono más oscuro al pasar el ratón.
 - "Guardar" tiene un fondo verde con un efecto de hover más oscuro, indicando la acción principal.

Editar contraseña

Nuevo valor para contraseña

Confirmar contraseña

Cancelar

Guardar

Editar email

Nuevo valor para email

Confirmar email

Cancelar

Guardar

- Archivo: PaymentModal.jsx
- Este componente es un modal de pago diseñado para permitir al usuario confirmar y realizar un pago ingresando los detalles de la tarjeta.
- Dependencias
 - PropTypes: Para la validación de las propiedades que recibe el componente.
 - motion (de framer-motion): Para aplicar animaciones al modal y mejorar la experiencia visual.
- Descripción del Componente
 - Nombre del Componente: PaymentModal
 - Props:
 - isOpen (bool, requerido): Indica si el modal está abierto.
 - onClose (función, requerido): Función que se ejecuta al hacer clic en el botón de cerrar o cancelar.
 - packageName (string, requerido): Nombre del paquete que el usuario está comprando.
 - price (number | string, requerido): Precio del paquete.
- Funcionalidades y Características
 - a. Renderización Condicional:
 - Si isOpen es false, el componente no se renderiza (return null).
 - Si isOpen es true, se muestra el modal en la pantalla.
 - b. Formulario de Pago:
 - El modal incluye campos para que el usuario ingrese el número de la tarjeta, nombre del propietario, fecha de expiración y CVV.
 - La información se almacena en el estado local del componente.
 - El formulario tiene validaciones simples como required para asegurar que todos los campos estén completos.
 - c. Lógica de Manejo de Pago:
 - Al enviar el formulario, la función handlePayment se ejecuta, simulando la confirmación del pago y marcando el estado isSubmitted como true para mostrar un mensaje de éxito.
 - Se puede agregar una lógica más compleja para conectar con una API de procesamiento de pagos.
 - d. Animaciones:
 - Utiliza motion de framer-motion para añadir efectos de aparición y desaparición suaves al modal, mejorando la experiencia visual.
 - La animación se basa en la opacidad y el escalado para una transición atractiva.
 - e. Interfaz de Usuario:
 - El modal tiene un diseño simple y moderno, con un fondo semitransparente que cubre toda la vista y un contenido centrado.
 - Incluye botones de acción para confirmar el pago o cancelar y cerrar el modal.
- Validación de Propiedades
 - PropTypes:
 - isOpen: Debe ser un booleano y es obligatorio.
 - onClose: Debe ser una función y es obligatoria.
 - packageName: Debe ser una cadena de texto y es obligatoria.
 - price: Puede ser un número o una cadena de texto y es obligatorio.
- Descripción Visual
 - **Modal:**
 - Aparece centrado en la pantalla con un fondo semitransparente que oscurece el resto de la vista.
 - Contenedor de fondo blanco con bordes redondeados y una sombra para dar profundidad.
 - Formulario de Pago:
 - Campos de entrada con bordes redondeados y cambios de color al enfocar.
 - Botones de confirmación y cancelación con colores distintivos y efectos de hover para indicar interactividad.
 - Animación:
 - Transición suave al mostrar y ocultar el modal para mejorar la experiencia del usuario.

- ProtectedRoute.jsx
 - Funcionalidad: Componente que protege las rutas de la aplicación, verificando si un usuario tiene permiso de acceso. Utiliza el contexto de autenticación para determinar si se debe permitir el acceso a la ruta o redirigir al inicio de sesión.
- AccountTypeSelect.jsx
 - Funcionalidad: Componente de selección que permite al usuario elegir un tipo de cuenta, como "Básica" o "Premium". Suele utilizarse en configuraciones de usuario o formularios de actualización de preferencias.
- AdminCard.jsx
 - Funcionalidad: Componente de tarjeta que se utiliza para mostrar información relacionada con los administradores. Puede incluir estadísticas o detalles importantes sobre la gestión de usuarios o contenido.
- CardEducation.jsx
 - Funcionalidad: Componente de tarjeta diseñado para presentar información educativa. Puede incluir recursos, artículos, o datos que fomenten la educación financiera o la gestión de recursos.
- CardMeta.jsx
 - Funcionalidad: Componente que muestra información sobre metas de usuario. Suele incluir detalles como el nombre de la meta, el progreso actual y los objetivos finales.
- CardReport.jsx
 - Funcionalidad: Componente de tarjeta que presenta un resumen de reportes generados por los usuarios. Ideal para secciones de análisis o gestión de información.
- CongratulationsModal.jsx
 - Funcionalidad: Modal que se utiliza para felicitar al usuario tras completar una acción importante, como alcanzar una meta o realizar un pago. Mejora la experiencia del usuario al agregar reconocimiento visual.
- Dialog.jsx
 - Funcionalidad: Componente genérico de dialogo que puede ser usado para mostrar mensajes de confirmación, alertas, o información adicional al usuario en una ventana emergente.
- EditableField.jsx
 - Funcionalidad: Campo de texto editable que permite al usuario modificar información directamente en el lugar donde se presenta, con la capacidad de guardar cambios localmente o enviarlos al backend.
- Footer.jsx
 - Funcionalidad: Pie de página de la aplicación que contiene enlaces, información legal y otros elementos secundarios, brindando contexto y accesos directos.
- Gastoslist.jsx
 - Funcionalidad: Lista de gastos del usuario, presentada de manera que se pueda explorar y filtrar fácilmente los diferentes elementos de gasto. Puede integrarse con gráficos y secciones de análisis.
- GlobalNotification.jsx
 - Funcionalidad: Sistema de notificaciones globales que se despliega para mostrar mensajes importantes al usuario, como actualizaciones, alertas de éxito o error.
- Header.jsx
 - Funcionalidad: Encabezado principal de la aplicación, comúnmente contiene el título de la sección actual y enlaces de navegación o información de perfil del usuario.
- HeaderAdmin.jsx
 - Funcionalidad: Versión del encabezado adaptada para la vista del administrador, incluyendo accesos rápidos a funciones de gestión y herramientas administrativas.

FrontEnd

- InformationCard.jsx
 - Funcionalidad: Tarjeta informativa que muestra datos relevantes sobre el usuario o la cuenta. Usada en paneles de control o en secciones de perfil para resumir información clave.
- Input.jsx
 - Funcionalidad: Componente de entrada reutilizable con estilos y validaciones predefinidas para su uso en formularios y otros elementos interactivos.
- Metaslist.jsx
 - Funcionalidad: Componente que muestra una lista de metas definidas por el usuario, facilitando la visualización y gestión de sus objetivos personales.
- Navbar.jsx
 - Funcionalidad: Barra de navegación superior que permite a los usuarios moverse fácilmente por la aplicación. Puede incluir iconos, enlaces y un menú de usuario.
- Notificacioneslist.jsx
 - Funcionalidad: Muestra una lista de notificaciones relevantes para el usuario, proporcionando detalles sobre acciones recientes o recordatorios importantes.
- RecentExpensesTable.jsx
 - Funcionalidad: Tabla que muestra los gastos recientes del usuario de manera clara y ordenada, facilitando el seguimiento de sus transacciones.
- Reminderslist.jsx
 - Funcionalidad: Componente que presenta una lista de recordatorios activos, ayudando al usuario a mantenerse organizado con sus tareas o eventos pendientes.
- ScrollArea.jsx
 - Funcionalidad: Componente de área de desplazamiento personalizado, diseñado para mostrar contenido largo de forma compacta y mejorando la experiencia de navegación dentro de secciones extensas.
- SelectAccountType.jsx
 - Funcionalidad: Elemento de selección utilizado para que el usuario elija un tipo de cuenta o plan en un formulario o proceso de registro.
- Sidebar.jsx
 - Funcionalidad: Barra lateral de navegación que ofrece accesos rápidos a diferentes secciones de la aplicación, mejorando la accesibilidad y organización del contenido.
- SidebarAdmin.jsx
 - Funcionalidad: Versión de la barra lateral adaptada para la vista de administrador, con accesos a funciones de gestión, herramientas y secciones exclusivas.
- Table.jsx
 - Funcionalidad: Componente de tabla reutilizable con estilos y formato predeterminado, empleado para presentar datos de manera estructurada en diferentes secciones de la aplicación.


```
import PropTypes from 'prop-types';

export function AdminCard({
  title,
  value = "N/A", // Valor predeterminado si no se proporciona `value`
  percentage,
  icon: Icon,
  color = "bg-gray-100",
  textColor = "text-black",
  loading = false, // Nuevo prop para controlar el estado de carga
  children,
}) {
  return (
    <div className={`bg-white rounded-lg shadow-lg p-6 ${color} ${textColor}`}>
      <div className="flex items-center justify-between mb-4">
        <h3 className="text-lg font-semibold">{title}</h3>
        {Icon && <Icon className="h-6 w-6" />}
      </div>
      <div className="text-3xl font-bold mb-2">
        {Loading ? "Cargando..." : value} {/* Mostrar "Cargando..." si está en estado de carga */}
      </div>
      {percentage !== undefined && !Loading && ( /* Ocultar el porcentaje mientras carga */
        <div className={`text-sm ${percentage > 0 ? 'text-green-500' : 'text-red-500'}`}>
          {percentage > 0 ? `+${percentage}%` : `-${percentage}%` }
        </div>
      )}
      {children}
    </div>
  );
}
```

```
import { useState } from 'react';

function Footer() {
  const [year, setYear] = useState(new Date().getFullYear());
  const [isHovered, setIsHovered] = useState(false);

  const className = "fixed bottom-0 left-0 w-full";
  const onMouseEnter = () => setIsHovered(true);
  const onMouseLeave = () => setIsHovered(false);

  /* Right-Aligned Footer */
  const isHovered = (
    <div className={`${text-gray-600} text-lg py-4 px-4 bg-white bg-opacity-70 shadow-sm absolute bottom-0 right-0`}
      <div className="text-right">
        {year}
      </div>
    </div>
  );

  /* Full-Width Footer */
  const footer = (
    <div className="text-right">
      {year}
    </div>
  );
  const className = "flex items-center justify-between px-8 py-6 bg-white bg-opacity-95 shadow-sm transition-all duration-300";
  const style = { height: isHovered ? 'auto' : '0', overflow: 'hidden' };

  <div className={`${flex justify-between w-full text-sm text-gray-600}`}
    <div className="text-right">
      {year}
    </div>
  </div>
}
```

```
import { NotificationContext } from './utils/NotificationContext';

export default function GlobalNotification() {
  const [notification, setNotification] = useState(null);
  const [showNotification, setShowNotification] = useState(false);

  if (!notification) return null;

  return (
    <div className="fixed bottom-4 right-4 flex flex-col items-end gap-4 z-50">
      <p>{notification}
      <button onClick={() => setNotification(null)}>Cerrar</button>
    </div>
  );
}
```

```
type={type}
placeholder={placeholder}
value={value}
onChange={onChange}
className={`border border-gray-300 rounded-lg px-4 py-2 focus:outline-none focus:border-blue-500`}
/>
);

input.propTypes = {
  type: PropTypes.string,
  placeholder: PropTypes.string,
  value: PropTypes.string,
  onChange: PropTypes.func,
  className: PropTypes.string,
}
```

Archive: **RewardCard.jsx**

Este archivo define un componente en React que permite a los usuarios canjear recompensas utilizando puntos acumulados. En este caso, se ofrece una recompensa para obtener una suscripción premium de 2 días. El componente muestra la cantidad de puntos requeridos, los puntos actuales del usuario y un botón para realizar el canje.

- Props:
 - `usuarioId` (number, requerido): El ID del usuario que está intentando canjear la recompensa.
 - `token` (string, requerido): Token de autenticación del usuario para las solicitudes de la API.
 - `puntosUsuario` (number, requerido): Los puntos actuales del usuario, utilizados para determinar si el usuario tiene suficientes puntos para canjear la recompensa.
 - `onRecompensaCanjeada` (func, requerido): Función que se ejecuta al canjear exitosamente la recompensa. Esta función se usa para actualizar los puntos o realizar otras acciones necesarias en el componente padre.

Funcionalidades y Características

- Estado interno:
 - `loading`: Indica si el canje de recompensa está en proceso.
 - `error`: Almacena cualquier mensaje de error durante el proceso de canje.
 - `success`: Indica si el canje de recompensa fue exitoso.
- Funciones:
 - `canjear Recompensa`:
 - Lógica para realizar el canje de recompensa. Verifica que el usuario tenga suficientes puntos.
 - Realiza una solicitud POST al backend para actualizar la suscripción del usuario.
 - Si la respuesta es exitosa, actualiza el estado de `success` y llama a `onRecompensaCanjeada` para actualizar los puntos en el componente padre.
 - Si ocurre un error, se muestra un mensaje de error.
- Renderización Condicional:
 - Si el usuario tiene menos de 100 puntos, se muestra un mensaje de error.
 - Si el canje es exitoso, se muestra un mensaje de éxito.
 - Se muestra un botón de canje que cambia su texto dependiendo de si la solicitud está en curso.

Interfaz de Usuario

- Card de Recompensa:
 - Una tarjeta con el diseño de una recompensa que incluye:
 - Una imagen en la parte superior que representa la recompensa.
 - Un título con el nombre de la recompensa.
 - Una breve descripción que explica que obtiene el usuario al canjear la recompensa.
 - Un área que muestra los puntos requeridos y los puntos actuales del usuario.
- Botón de Canje:
 - Un botón que permite al usuario canjear la recompensa. El texto del botón cambia dependiendo del estado de la solicitud (cargando o listo para canjear).
 - Si el usuario tiene suficientes puntos, puede hacer clic en el botón para iniciar el canje.



Archivo: Metaslist.jsx

Este archivo define un componente en React que permite gestionar una lista de metas financieras, permitiendo a los usuarios realizar diversas acciones como añadir montos, editar metas, eliminar metas y ver los puntos ganados por completar metas. (cargando o listo para canjear).

Funcionalidades del componente Metaslist

- **Mostrar metas:** Muestra una lista de metas con la capacidad de paginarlas. Cada meta tiene un objetivo monetario y un monto actual. Los usuarios pueden interactuar con estas metas mediante varios modales.
- **Agregar monto:** Permite a los usuarios añadir un monto adicional a una meta, y si esta se cumple, el sistema la marca como completada y le otorga puntos al usuario.
- **Editar meta:** Los usuarios pueden modificar la información de una meta existente.
- **Eliminar meta:** Los usuarios pueden eliminar metas, aunque esto requiere una confirmación.
- **Paginación:** El componente permite navegar entre las metas mediante paginación.
- **Modales:** Usa modales para varias interacciones:
 - **AddAmountModal:** Para agregar montos a las metas.
 - **EditGoalModal:** Para editar las metas.
 - **SuccessModal:** Para mostrar un mensaje de éxito con los puntos ganados.
 - **CongratulationsModal:** Para felicitar al usuario por completar una meta.

Estado del Componente

- **metas:** Arreglo que contiene las metas obtenidas desde la API.
- **selectedMeta:** Objeto que representa la meta seleccionada para realizar acciones (editar, añadir monto, etc.).
- **isAddAmountModalOpen:** Booleano que indica si el modal para añadir monto está abierto.
- **isEditModalOpen:** Booleano que indica si el modal para editar la meta está abierto.
- **isDeleteModalOpen:** Booleano que indica si el modal de confirmación para eliminar la meta está abierto.
- **metaToDelete:** Objeto que representa la meta seleccionada para eliminar.
- **isCongratulationsModalOpen:** Booleano que indica si el modal de felicitaciones está abierto.
- **isSuccessModalOpen:** Booleano que indica si el modal de éxito (con los puntos ganados) está abierto.

Comportamiento del Componente CardMeta

- **title:** Muestra el nombre de la meta.
- **monto_actual:** Muestra el monto actual de la meta (con formato de moneda).
- **monto_objetivo:** Muestra el monto objetivo de la meta (con formato de moneda).
- **nombre_categoria:** Muestra el nombre de la categoría asociada con la meta.
- **actions:** Permite acciones sobre la meta (añadir monto, editar, eliminar).

Comportamiento de los Modales

- **AddAmountModal:** Permite al usuario añadir un monto adicional a la meta seleccionada.
 - **montoActual:** El monto actual de la meta se pasa como prop al modal.
 - **onSave:** Función que guarda el monto añadido y actualiza la meta.
- **EditGoalModal:** Permite al usuario editar los detalles de la meta seleccionada.
 - **meta:** Los datos de la meta se pasan como prop al modal.
 - **onSave:** Función que guarda los cambios realizados en la meta.
- **CongratulationsModal:** Muestra un mensaje de felicitación cuando la meta es completada.
 - **meta:** La meta completada se pasa como prop al modal.
- **SuccessModal:** Muestra los puntos ganados al completar una meta.
 - **points:** Los puntos obtenidos se pasan como prop al modal.

Archivo: **RewardsPage.jsx**

Este componente de React representa la página de recompensas del usuario, donde pueden ver canjear las recompensas disponibles en función de los puntos que han acumulado. La página muestra un listado de tarjetas de recompensas con información sobre cada recompensa, y permite al usuario canjear las recompensas usando sus puntos.

Estado del Componente

- **usuarioid**: Almacena el ID del usuario, extraído del token JWT. Es utilizado para hacer la solicitud de puntos.
- **token**: Almacena el token JWT que se obtiene desde localStorage para hacer solicitudes autenticadas al backend.
- **puntosUsuario**: Almacena los puntos acumulados por el usuario. Es obtenido del backend y usado para mostrar el saldo actual de puntos y verificar si el usuario puede canjear recompensas.

Funcionalidades Principales

Obtener puntos del usuario

- **Token JWT**: Se extrae el token JWT del localStorage, el cual es necesario para autenticar la solicitud.
- **parseJwt**: Esta función decodifica el token y extrae el ID del usuario para realizar la solicitud de puntos.
- **Solicitud a la API**: Se hace una solicitud GET a la API para obtener los puntos del usuario utilizando su **usuarioid**. Los puntos se almacenan en el estado **puntosUsuario** para ser utilizados más adelante.

Canjear recompensas

- Al canjear una recompensa, los puntos del usuario se reducen en 100 (el costo fijo por recompensa). Esto se maneja a través de la función **handleRecompensaCanjeada**, que actualiza el estado **puntosUsuario**.

Renderizado de recompensas

- Las recompensas se muestran utilizando el componente **RewardCard**, el cual recibe la información del usuario (ID, token y puntos disponibles) y permite al usuario canjear recompensas.

Comportamiento del Componente **RewardCard**

- **usuarioid**: El ID del usuario, que se pasa como prop para poder realizar las acciones sobre las recompensas.
- **token**: El token JWT del usuario, utilizado para autenticar las solicitudes.
- **puntosUsuario**: Los puntos acumulados por el usuario, que se utilizan para verificar si tiene suficiente saldo para canjear una recompensa.
- **onRecompensaCanjeada**: Función que se ejecuta cuando el usuario canjea una recompensa, la cual actualiza el saldo de puntos.

Renderizado del Componente

El componente **RewardsPage** renderiza lo siguiente:

1. **Sidebar**: La barra lateral de navegación que permite al usuario navegar a otras páginas.
2. **Header**: El encabezado de la página con el título "Pagina de Recompensas".
3. **Grid de recompensas**: Un grid de tarjetas de recompensas, donde cada tarjeta muestra información de una recompensa específica (nombre, descripción, puntos requeridos).
4. **Cargando**: Si el **usuarioid** no está disponible, se muestra un mensaje de "Cargando..." mientras se obtiene el ID del usuario y sus puntos.