FRONTEND

El archivo inicial es el archivo `index.js` que se encuentra en la ruta `/src` del proyecto. Este es el punto donde se crea el router de `react-router-dom` que administra las rutas de la aplicación.

Para agregar una ruta nueva, se coloca dentro del componente `Routes`, creando un nuevo componente `Route` como se muestra en el siguiente código:

```
<Route path="/mi-nueva-ruta" element={<MiNuevoComponente />} />
```

En este ejemplo se agrega una nueva ruta "/mi-nueva-ruta" que renderiza el componente `MiNuevoComponente`.

Los componentes de las rutas se encuentran en la carpeta src/pages/

Pagina principal (componente App.js)

Este componente se encarga de renderizar el contenido de la página principal de la aplicación.

Importaciones

- Se importa el componente `Nav` desde la ruta `../components/Nav`.

Estructura de la página

- Se utiliza JSX para especificar la estructura de la página.
- Se utiliza el componente `Nav`, pasando como propiedad `typeNav` el valor "index".
- Se crea un contenedor con una clase de Bootstrap `bg-secondary p-5 text-center shadow-sm` y un estilo personalizado `style={{ "--bs-bg-opacity": 0.2 }}`.
- Dentro del contenedor se crean dos elementos de tipo `h2` y `p` con alineación de texto `justify`.
- Se crea un segundo contenedor con una clase de Bootstrap `d-flex col-12 p-5 navbar-nav`.
- Dentro del segundo contenedor se crea un elemento de tipo `p` con alineación de texto `justify`.

Exportación

- Se exporta el componente `App` para poder ser utilizado en otras partes de la aplicación.

Componente Login

Este componente se encarga de renderizar la página de inicio de sesión de la aplicación.

Importaciones

- Se importa el componente `FormLogin` desde la ruta `../components/Login/formLogin`.
- Se importa el componente `Nav` desde la ruta `../components/Nav`.
- Se importa la función `loginAPI` desde la ruta `../utils/sign/login`.

Funcionalidad

- Se utiliza el hook `React.useState` para crear tres estados: `user`, `password`, y `espera`.
- Se crea una función `login` que se encarga de gestionar el inicio de sesión.
- La función recibe un evento de tipo `submit`, previene su comportamiento por defecto, cambia el estado de `espera` a `true` y llama a la función `loginAPI` importada, pasándole los valores del estado `user` y `password`.
- Se utiliza `setTimeout` para cambiar el estado de `espera` a `false` después de 2 segundos.
- Se utiliza JSX para especificar la estructura de la página.
- Se utiliza el componente `Nav`, pasando como propiedad `typeNav` el valor "index".
- Se utiliza el componente `FormLogin` importado, pasando como props la función `login`, y las funciones `setUser` y `setPassword` para actualizar los estados respectivos.
- Se utiliza una condicional para mostrar un spinner mientras se está realizando el inicio de sesión.

Exportación

- Se exporta el componente `Login` para poder ser utilizado en otras partes de la aplicación.

Componente FormLogin

Este componente se encarga de renderizar el formulario de inicio de sesión dentro del componente `Login`.

Props

- Se recibe como propiedad la función `login` que se ejecuta al hacer submit en el formulario.
- Se recibe como propiedad las funciones `setUser` y `setPassword` para actualizar los estados respectivos en el componente `Login`.

Funcionalidad

- Se utiliza el hook `React.useState` para crear un estado `mostrarPWD` con valor inicial "password".

- Se crea una función `mostrarPass` que se encarga de cambiar el estado `mostrarPWD` a "text" o "password" dependiendo su valor actual.
- Se utiliza JSX para especificar la estructura del formulario.
- Se utiliza la etiqueta `form` con una clase de Bootstrap `d-flex flex-column p-5 align-items-center` y un evento `onSubmit` que ejecuta la función `login`.
- Dentro del formulario se utilizan dos etiquetas `input` con una clase de Bootstrap `form-control mb-2 text-center` y placeholder "USUARIO" y "CONTRASEÑA" respectivamente.
- Estos inputs también tienen un evento `onChange` que ejecuta las funciones `setUser` y `setPassword` respectivamente, para actualizar los estados `user` y `password` en el componente `Login`.
- Se utiliza un botón con una clase de Bootstrap `btn col-12 mb-2` y un evento `onClick` que ejecuta la función `mostrarPass`.
- Se utiliza un segundo botón con una clase de Bootstrap `btn btn-primary col-12 mb-2` y un evento `onSubmit` que ejecuta la función `login`.
- Se utiliza una etiqueta `Link` de `react-router-dom` con una clase de Bootstrap `col-12` y una dirección `to='/sign-up'` que lleva al componente de registro.En resumen, este componente se encarga de mostrar el formulario de inicio de sesión con dos inputs para ingresar el usuario y contraseña, un botón para mostrar/ocultar la contraseña y un botón para iniciar sesión, también tiene un enlace para registrarse.

Componente Signup

Este componente se encarga de renderizar la página de registro de nuevos usuarios de la aplicación.

Importaciones

- Se importa el componente `Nav` desde la ruta `../components/Nav`.
- Se importa el componente `Advertencia` desde la ruta `../components/views/Admin/Signup/Advertencia`
- Se importa el componente `FormularioRegistro` desde la ruta `../components/views/Admin/Signup/FormularioRegistro`

Funcionalidad

- Se utiliza el hook `React.useState` para crear un estado `ok`
- Se utiliza una condicional para mostrar el componente `Advertencia` si el estado `ok` es igual a `false` o el componente `FormularioRegistro` si el estado `ok` es igual a `true`
- Se utiliza el componente `Nav`, pasando como propiedad `typeNav` el valor "index".

Exportación

- Se exporta el componente `Signup` para poder ser utilizado en otras partes de la aplicación.

Componente Advertencia

Este componente se encarga de mostrar una advertencia antes de continuar con el proceso de registro.

Funcionalidad

- Se utiliza JSX para especificar la estructura de la advertencia.
- Se utiliza una lista ordenada para mostrar las características necesarias para continuar con el registro.
- Se utiliza un botón con un evento `onClick` que ejecuta la función `ok` recibida como propiedad, cambiando el estado de `ok` a `true`.

Exportación

- Se exporta el componente `Advertencia` para poder ser utilizado en otras partes de la aplicación.

Componente FormularioRegistro

Este componente se encarga de renderizar el formulario de registro de la aplicación.

Importaciones

- Se importa el componente `Link` desde `react-router-dom`.
- Se importa la función `CrearUsuario` desde la ruta `../../../../utils/sign/Registro`.

Funcionalidad

- Se crea una función `guardar` que se encarga de gestionar el registro del usuario.
- La función recibe un evento de tipo `submit`, previene su comportamiento por defecto, y utiliza los valores de los inputs para crear un objeto `obj` con toda la información del usuario.
- Se utiliza la función `CrearUsuario` importada para enviar el objeto y crear el usuario en la base de datos.
- Se utiliza JSX para especificar la estructura del formulario de registro.
- El formulario cuenta con varios inputs de texto, un input de tipo fecha, un select y un botón para enviar la información.

Exportación

- Se exporta el componente `FormularioRegistro` para poder ser utilizado en otras partes de la aplicación.

Componente Home

Este componente se encarga de renderizar la página principal de la aplicación para usuarios registrados.

Importaciones

- Se importa el componente `Nav` desde la ruta `../components/Nav`.
- Se importa el componente `HomeView` desde la ruta `../components/views/Client/HomeView`.

Funcionalidad

- Se utiliza JSX para especificar la estructura de la página.
- Se utiliza el componente `Nav`, pasando como propiedad `typeNav` el valor "user" para especificar que se trata de una página para usuarios registrados.
- Se utiliza el componente `HomeView` importado para mostrar la vista principal de la página.

Exportación

- Se exporta el componente `Home` para poder ser utilizado en otras partes de la aplicación.

Componente HomeView

Este componente se encarga de mostrar la vista principal de la página principal de la aplicación para usuarios registrados.

Importaciones

- Se importa la constante `API_BC_REPORTE` desde la ruta `../../../utils/Constants`
- Se importa el componente `TipsList` desde la ruta `./TipList`

Funcionalidad

  - Se utiliza el hook `useState` para crear un estado llamado `reporteDatos` que almacena los datos del historial crediticio del usuario.
  - Se utiliza el hook `useEffect` para realizar una petición `GET` a la API especificada en `API_BC_REPORTE` y actualizar el estado `reporteDatos` con la respuesta.
  - Se utiliza una función `nuevaConsulta` para redirigir al usuario a la página de métodos de pago.

- Se utiliza JSX para especificar la estructura de la página.
- Se utiliza una condicional para mostrar un mensaje de alerta dependiendo del puntaje del historial crediticio del usuario.
- Se utiliza una tabla para mostrar los datos del reporte, incluyendo el puntaje, el número de cuentas y la fecha de la última solicitud de reporte.
- Se utiliza el componente TipsList para mostrar consejos sobre cómo mejorar el historial crediticio.

Componente TipsList

Este componente se encarga de mostrar los consejos sobre cómo mejorar el historial crediticio.

Importaciones

- Se importa la librería `Swal` desde `sweetalert2`, la cual se utiliza para mostrar una ventana emergente con información sobre el consejo seleccionado.
- Se importa el archivo `CompelteTips` desde `../../../utils/Client/tips` el cual contiene los consejos para ser mostrados.

Funcionalidad

- Se utiliza el estado `score` para filtrar los consejos a mostrar.
- Se utiliza una función `showTip` para mostrar una ventana emergente con información sobre el consejo seleccionado.
- Se utiliza JSX para especificar la estructura de la lista de consejos.
- Se utiliza un `map` para recorrer el arreglo de consejos y mostrarlos en una lista de tarjetas.
- Se utiliza un evento `onClick` en cada tarjeta para mostrar la ventana emergente con información sobre el consejo seleccionado.

Exportación

- Se exporta el componente `TipsList` para poder ser utilizado en otras partes de la aplicación.

Componente Admin

Este componente se encarga de renderizar la página principal de la aplicación para usuarios administradores.

Importaciones

- Se importa el componente `AdminMenu` desde la ruta `../components/Admin/AdminMenu`.
- Se importa el componente `Nav` desde la ruta `../components/Nav`.
- Se importa el componente `Tarjetas` desde la ruta `../components/views/Admin/Tarjetas`.
- Se importa el componente `Usuarios` desde la ruta `../components/views/Admin/Usuarios`.

Funcionalidad

- Se utiliza el estado `pagina` para controlar qué vista se muestra en el componente.
- Se utiliza el componente `AdminMenu` para mostrar un menú que permite al usuario administrador cambiar entre las diferentes vistas.
- Se utiliza el componente `Nav` para mostrar la barra de navegación en la página.
- Se utiliza un condicional para mostrar la vista correspondiente (`Usuarios` o `Tarjetas`) dependiendo del valor de `pagina`.

Exportación

- Se exporta el componente `Admin` para poder ser utilizado en otras partes de la aplicación.

Componente AdminMenu

Este componente se encarga de mostrar el menú de opciones para el administrador.

Importaciones

- No se importan librerías ni archivos externos.

Funcionalidad

- Se utiliza el estado `pagina` para determinar qué opción del menú está seleccionada.
- Se utiliza el estado `setPagina` para cambiar la opción seleccionada en el menú.
- Se utiliza JSX para especificar la estructura del menú.
- Se utiliza un `if` para determinar qué opción del menú debe mostrarse como activa.
- Se utiliza un evento `onClick` en cada opción del menú para cambiar la opción seleccionada.

Exportación

- Se exporta el componente `AdminMenu` para poder ser utilizado en otras partes de la aplicación.

Componente Usuarios

Este componente se encarga de mostrar una lista de usuarios y permitir buscar a un usuario específico.

Importaciones

- Se importa el componente `Search` desde `./Usuarios/Search`, el cual se encarga de la búsqueda de un usuario específico.
- Se importa el componente `TablaUsuarios` desde `./Usuarios/TablaUsuarios`, el cual se encarga de mostrar la lista de usuarios.

Funcionalidad

- Se utiliza el estado `personas` para almacenar la lista de usuarios a mostrar.
- Se utiliza el componente `Search` para permitir buscar a un usuario específico.
- Se utiliza el componente `TablaUsuarios` para mostrar la lista de usuarios.
- Se utiliza JSX para especificar la estructura de la página.

Exportación

- Se exporta el componente `Usuarios` para poder ser utilizado en otras partes de la aplicación.

Componente Tarjetas

Este componente es el encargado de mostrar las tarjetas y los métodos de pago registrados en la aplicación.

Importaciones

- Se importa el componente `TablaTarjetas` desde `./Tarjetas/TablaTarjetas` el cual muestra una tabla con información de las tarjetas registradas.
- Se importa el componente `FormularioTarjetas` desde `./Tarjetas/FormularioTarjetas` el cual permite agregar o editar una tarjeta registrada.
- Se importa el componente `TablaMetodos` desde `./MetodosPago/TablaMetodos` el cual muestra una tabla con información de los métodos de pago registrados.
- Se importa el componente `FormularioMetodos` desde `./MetodosPago/FormuarioMetodos` el cual permite agregar o editar un método de pago registrado.

Funcionalidad

- El componente `Tarjetas` se encarga de mostrar dos secciones: una para las tarjetas y otra para los métodos de pago.
- En la sección de tarjetas, se importa y se utiliza el componente `TablaTarjetas` para mostrar una tabla con información sobre las tarjetas registradas. También se importa y se utiliza el componente `FormularioTarjetas` para permitir al usuario agregar o editar información sobre las tarjetas.
- En la sección de métodos de pago, se importa y se utiliza el componente `TablaMetodos` para mostrar una tabla con información sobre los métodos de pago registrados. También se importa y se utiliza el componente `FormularioMetodos` para permitir al usuario agregar o editar información sobre los métodos de pago.
- El componente utiliza JSX para especificar la estructura de las dos secciones y los componentes importados.

Exportación

- Se exporta el componente `Tarjetas` para poder ser utilizado en otras partes de la aplicación.

Componente Pagos

Este componente se encarga de mostrar los metodos de pagos disponibles para el usuario.

Importaciones

- Se importa el componente Nav desde `../components/Nav` , el cual se utiliza para mostrar la barra de navegación en la página.
- Se importa la constante `API_BC_METODOS` desde `../utils/Constants` , la cual se utiliza para hacer una petición al servidor para obtener los metodos de pagos.

Funcionalidad

- Se utiliza el hook `useEffect` para hacer una petición al servidor para obtener los metodos de pagos disponibles una vez que el componente se monta.
- Se utiliza el hook `useState` para almacenar los metodos de pagos en un estado.
- Se utiliza un `map` para recorrer el arreglo de metodos de pagos y mostrarlos como botones en la página.
- Se utiliza un evento `onClick` en cada botón para redirigir al usuario a la página de detalles del metodo de pago seleccionado.
- Se utiliza la función `continuar` para redirigir al usuario a la página de detalles del metodo de pago seleccionado, se construye la url con el nombre del metodo de pago y el id del mismo.

Exportación

- Se exporta el componente `Pagos` para poder ser utilizado en otras partes de la aplicación.

Backend

Este proyecto

Se comienza por el archivo `Index.js` que se encuentra en la raíz del proyecto, aquí se llama la función para crear el servidor que se encuentra en `/src/configs/server` y se agregan las rutas de express para las consultas a la API.

Frontend

El proyecto frontend se agrega en la carpeta `public` donde el servidor express lo ejecuta en el puerto definido (3001).

El codigo se encuentra en la carpeta `/src` , esta se divide en las siguientes carpetas:

Carpeta routes

Donde se definen las rutas para cada acción de la API.

Carpeta controllers

Donde se establecen los métodos de entrada de cada ruta y método definidos.

Carpeta database

Donde se hace la comunicación con la base de datos de la API.

Carpeta configs

Donde se definen características de la aplicación, como credenciales, tokens y la configuración inicial del servidor.

Carpeta utils

Donde se definen operaciones importantes para las consultas, por ejemplo la validacion de los tokens.

API

El archivo `index.js` contiene las siguientes rutas extraidas de la carpeta `routes` :

```
app.use("/v1/user/", user);
app.use("/v1/card/", card);
app.use("/v1/payment/", method);
app.use("/v1/moffin/report", moffin);
```

Estas rutas se usan para establecer la URL base para cada una de las funciones de la API. Por ejemplo, `/v1/user/` será la URL base para todas las acciones relacionadas con el usuario, mientras que `/v1/card/` será la URL base para las acciones relacionadas con las tarjetas, y así sucesivamente.

Cada ruta tiene un archivo en la carpeta `routes` donde se definen las operaciones que tendra disponible, las operaciones que se pueden aplicar son:

- **POST**: Para ingresar datos.
- **GET**: Para obtener datos.
- **PATCH/PUT**: Para modificar datos.
- **DELETE**: Para eliminar datos.

Ejemplo:

```
const user = require("express").Router();
const UserRoute = require("../controllers/user");

// create user
user.post("/", UserRoute.create);
```

En la carpeta controllers se crean los metodos de entrada, con dos parametros `req` y `res` para recibir y mandar datos de las peticiones.

Tambien en estos metodos se hace la validacion de los tokens de los usuarios mandados en la solicitud para habilitar o restringir ciertas operaciones para los distintos tipos de usuario y saber si aun tienen una sesion activa.

un ejemplo es:

```
exports.delete = (req, res) => {
    const id = req.params.id;
```

```
    const token = req.headers["authorization"];
    if (validateToken(token, keyAccess)) {
        //Operacion para eliminar en la base de datos
    } else {
        res.status(403).json({ message: "Access denied, invalid token" });
    }
};
```

En la carpeta `database` se definen dentro de carpetas como por ejemplo `User` para los usuarios la operacion correspondiente con el `nombre_de_la_carpeta.model.js`

```
User.delete = (id, callback) => {
    console.log(id);
    database.query(`DELETE FROM persona WHERE idPersona = '${id}'`, (err) => {
        if (err) {
            callback(err, { delete: false });
        } else {
            callback(null, { delete: true });
        }
    });
};
```