# Prueba Técnica – Sistema Bancario

## Descripción de proyecto:

El objetivo del proyecto es desarrollar una aplicación WEB que permita realizar operaciones CRUD (Create, read, update, delete), la aplicación WEB debe contar con autenticación del usuario por medio de cookies o variables de sesión, la aplicación WEB deberá estar en un repositorio de GitHub o Bitbucket (preferiblemente).

Crear una aplicación WEB que permita el ingreso a un sistema bancario, así como realizar operaciones transaccionales básicas. Deberá haber dos tipos de perfiles uno admin y otro cliente, donde el admin podrá ver un reporte de las transacciones generadas y el cliente podrá realizar retiros, depósitos y transferencias entre cuentas además podrá ver las transacciones que ha realizado.

# Requisitos

## Requisitos funcionales:

### Crear usuario:

* El sistema deberá solicitar el tipo de usuario que se va a crear, ya sea admin o cliente.
* El sistema debe solicitar: Nombre, identificación, correo electrónico, contraseña.
* Validar que se escriban todos los campos.
* Validar que sea un correo electrónico válido.
* Validar que la contraseña tenga letras y números.
* Validar que el número de identificación y correo no existan en la base de datos.
* El sistema asignará un número aleatorio de 10 cifras el cual será el número de cuenta.
* El cliente iniciará con un monto de $1000.000.

### Login:

* El sistema permitirá al usuario loquearse con su cuenta.
* Validar correo electrónico del cliente.
* Validar contraseña del cliente.

### Control de errores de inserciones, updates y demás.

## Requisitos No Funcionales:

* Usar motor de base de datos SQL Server o PostgreSQL.
* Se recomienda que separe los componentes del backend y del frontend, se sugiere que use frameworks como ScriptCase, Java, Spring Boot, Node.js o VB.net.
* Para el versionamiento de código puede usar SourceTree.

# Desarrollo

## Requisitos no funcionales:

Se eligió postgresql para la elaboración de la base de datos, por su versatilidad para el manejo, los componentes fueron separados en FrontEnd (Angular) y BackEnd (Node JS).

## Requisitos funcionales:

### Diagrama entidad relación

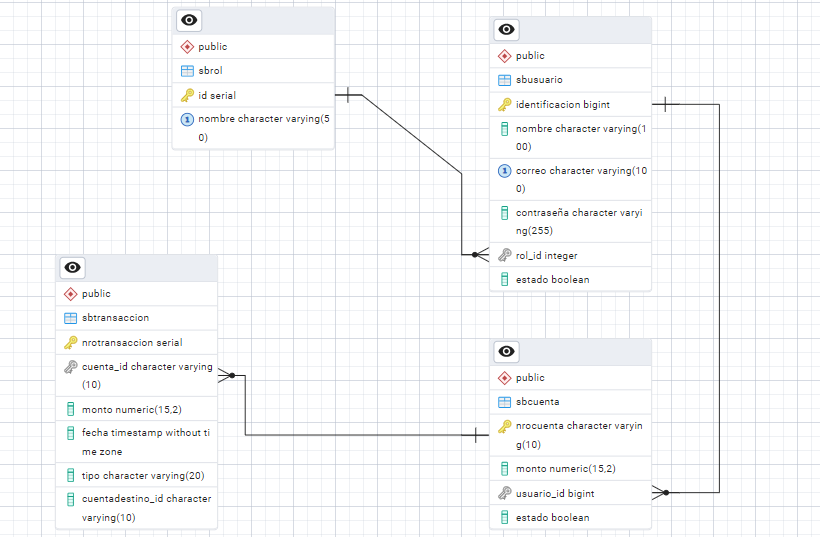
|  |  |  |
| --- | --- | --- |
| sbcuenta | | |
| Contiene información sobre las cuentas bancarias de los usuarios. | | |
| Columnas | **Tipo** | **Descripción** |
| nrocuenta | character varying(10) | Número de cuenta, clave primaria. |
| monto | numeric(15,2) | Monto disponible en la cuenta. |
| usuario\_id | bigint | Identificación del usuario (FK de sbusuario). |
| estado | boolean | Estado de la cuenta (activo/inactivo), por defecto true. |

|  |  |  |
| --- | --- | --- |
| sbrol | | |
| Registra las transacciones de las cuentas bancarias. | | |
| Columnas | **Tipo** | **Descripción** |
| id | integer | Identificador del rol, clave primaria. |
| nombre | character varying(50) | Nombre del rol. |

|  |  |  |
| --- | --- | --- |
| sbtransaccion | | |
| Registra las transacciones de las cuentas bancarias. | | |
| Columnas | **Tipo** | **Descripción** |
| nrotransaccion | integer | Número de la transacción, clave primaria. |
| cuenta\_id | character varying(10) | Número de cuenta involucrada (FK de sbcuenta). |
| monto | numeric(15,2) | Monto de la transacción. |
| fecha | timestamp | Fecha y hora de la transacción, valor por defecto CURRENT\_TIMESTAMP. |
| tipo | character varying(20) | Tipo de transacción (depósito, retiro, transferencia). |
| cuentadestino\_id | character varying(10) | Número de cuenta de destino (en caso de transferencia). |

|  |  |  |
| --- | --- | --- |
| sbusuario | | |
| Almacena la información de los usuarios del sistema. | | |
| Columnas | **Tipo** | **Descripción** |
| identificacion | bigint | Número de identificación del usuario, clave primaria. |
| nombre | character varying(100) | Nombre del usuario. |
| correo | character varying(100) | Correo electrónico del usuario, valor único. |
| contraseña | character varying(255) | Contraseña cifrada del usuario. |
| rol\_id | integer | ID del rol del usuario (FK de sbrol). |
| estado | boolean | Estado del usuario (activo/inactivo), por defecto true |

### Diagrama entidad-relación



## BackEnd

Se desarrolló una API RESTful, estructurada en cuatro componentes: controllers, middlewares, models y routes, cada uno encargado de una parte importante dentro de la misma. Esta arquitectura permite una gestión eficiente y escalable del código, facilitando el mantenimiento y la extensión de la API.

En la implementación del servidor, se utiliza Express, que se configura para manejar las rutas de autenticación y transacciones. La API se establece en el puerto definido en las variables de entorno mediante el uso del archivo `.env` (3000), que permite gestionar de forma segura las configuraciones sensibles, como las credenciales de conexión a la base de datos. Además, se incorpora un middleware para procesar datos en formato JSON y otro para habilitar CORS, que permite el acceso controlado a recursos entre diferentes dominios. También se incluye un middleware genérico para manejar errores, que devuelve una respuesta adecuada en caso de que se produzca un problema interno del servidor.

En esta API RESTful, los controllers desempeñan un papel fundamental en la gestión de la lógica de negocio y la interacción con la base de datos. Cada controller se encarga de manejar las solicitudes entrantes, procesar los datos y devolver las respuestas adecuadas al cliente. En el contexto de esta API, los controllers están organizados para gestionar diferentes entidades, como la autenticación de usuarios y la manipulación de transacciones. Por ejemplo, el controller de autenticación verifica las credenciales del usuario, y maneja el registro y la edición de usuarios. Esta separación de la lógica de negocio en controllers facilita el mantenimiento del código, promueve la reutilización de funciones y mejora la legibilidad del proyecto, permitiendo que los desarrolladores se concentren en implementar nuevas características y mejoras.

Dentro de los middlewares, se encuentra authenticateToken, que se encarga de verificar la autenticidad de un token JWT (JSON Web Token) en las solicitudes a la API. Este middleware extrae el token del encabezado de autorización, verifica su existencia y validez, y asegura que solo las solicitudes con un token válido puedan acceder a las rutas protegidas. Si el token es válido, se añade la información del usuario al objeto de solicitud para que los controladores posteriores puedan acceder a ella.

El authController, responsable de la autenticación de usuarios, incluye funcionalidades como el registro, el inicio de sesión, la edición de usuarios y la activación o desactivación de cuentas. Este controlador se basa en la validación de datos y en la encriptación de contraseñas, asegurando que las contraseñas sean almacenadas de manera segura en la base de datos. En el proceso de inicio de sesión, se valida la existencia del correo electrónico y la correspondencia de la contraseña, generando un token JWT que se devuelve al usuario para su autenticación en futuras solicitudes.

El apartado models posee el archivo db.js, el cual se encarga de gestionar la conexión a la base de datos PostgreSQL mediante el módulo pg, que permite interactuar con bases de datos en Node.js. En primer lugar, se importa el módulo Pool de pg para crear un grupo de conexiones y se requiere el paquete dotenv, que permite cargar variables de entorno desde un archivo .env. Luego, se crea una instancia de Pool utilizando las credenciales de la base de datos almacenadas en las variables de entorno, lo que incluye el usuario, el host, el nombre de la base de datos, la contraseña y el puerto. Este enfoque asegura que la información sensible no esté hardcodeada en el código. Además, se incluye un bloque condicional que verifica si la conexión se ha establecido correctamente, mostrando un mensaje en la consola que indica "conexión exitosa". Finalmente, se exporta un objeto que contiene una función query, la cual permite realizar consultas a la base de datos, recibiendo como parámetros el texto de la consulta SQL y los valores a sustituir. Esta abstracción simplifica la interacción con la base de datos desde los controllers, permitiendo que las consultas se realicen de manera más clara y organizada. Esta estructura facilita la gestión de la conexión a la base de datos, garantizando un acceso eficiente y seguro a los datos, y promoviendo la mantenibilidad y escalabilidad del código al separar la lógica de la base de datos del resto de la aplicación.