Resumen Detallado del Backend Desarrollado Hasta Ahora

Este es el resumen detallado del backend que hemos desarrollado para la plataforma de Impulso Capital, donde se manejan usuarios, roles y permisos, asegurando que el sistema sea modular, escalable y seguro.

---

1. Configuración Inicial

Estructura del Proyecto

El proyecto backend se ha organizado dentro de la carpeta `back`. Utilizamos **Node.js**, **Express** como framework para el servidor, **Sequelize** como ORM y **PostgreSQL** como la base de datos relacional. A continuación, detallamos los principales archivos y directorios que estructuran el backend:

- **`server.js`**: Punto de entrada del servidor, donde se configuran las rutas y se inicializa la conexión con la base de datos.
- **`src/models/`**: Contiene los modelos Sequelize para las entidades `User`, `Role`, `Permission`, y las asociaciones entre ellas.
- **`src/routes/`**: Define las rutas de la API para usuarios, roles y permisos.
- **`src/controllers/`**: Lógica de control para gestionar las operaciones CRUD de usuarios, roles, y permisos.
- **`src/middlewares/`**: Autenticación JWT y autorización basada en roles y permisos.

---

2. Base de Datos y Modelos

Modelos Definidos

- **`User.js`**: Modelo que representa a los usuarios del sistema.
  - Campos: `id`, `username`, `email`, `password`, `role_id`.
  - Relaciones: Cada usuario tiene un rol a través del campo `role_id`.
- **`Role.js`**: Modelo que define los roles de los usuarios.
  - Campos: `id`, `role_name`, `description`.
  - Relaciones: Los roles tienen una relación muchos a muchos con los permisos.
- **`Permission.js`**: Modelo que define los permisos del sistema.
  - Campos: `id`, `permission_name`.
  - Relaciones: Los permisos tienen una relación muchos a muchos con los roles.

Asociaciones (relaciones) entre modelos

- Los roles y permisos tienen una **relación muchos a muchos**, representada por la tabla intermedia `role_permissions`.
- Cada usuario tiene un **rol único** definido por la relación con el modelo `Role`.

---

3. Autenticación y Autorización

Autenticación con JWT

- Utilizamos **JWT (JSON Web Tokens)** para autenticar a los usuarios.
- Durante el inicio de sesión, se genera un token JWT que contiene el `id` del usuario, su `email`, y su `role_id`.
- El token tiene una duración de 1 hora y debe ser enviado en el encabezado `Authorization` en todas las solicitudes protegidas.

Autorización por Roles y Permisos

- Implementamos un middleware `authenticateRole` para verificar si el usuario tiene el rol adecuado para acceder a determinadas rutas.
- También creamos el middleware `authorizePermission` para verificar si el usuario tiene el **permiso** necesario para acceder a una ruta específica.

Ejemplo de Rutas Protegidas

- **Ruta para obtener todos los usuarios**: Protegida con el permiso `view_users`.
- **Ruta para crear un nuevo usuario**: Protegida con el permiso `manage_users`.

---

4. Funcionalidades CRUD Implementadas

1. Usuarios (User)

- **Crear usuario (`POST /api/users`)**: Permite registrar un nuevo usuario. Está protegida por el permiso `manage_users`.
- **Obtener usuarios (`GET /api/users`)**: Devuelve la lista de todos los usuarios. Solo accesible por usuarios con el permiso `view_users`.
- **Actualizar usuario (`PUT /api/users/:id`)**: Permite modificar los datos de un usuario existente, protegido por `manage_users`.
- **Eliminar usuario (`DELETE /api/users/:id`)**: Permite eliminar un usuario, solo accesible por usuarios con el permiso `manage_users`.

2. Roles (Role)

- **Crear rol (`POST /api/roles`)**: Permite crear un nuevo rol en el sistema.

- **Obtener roles (`GET /api/roles`)**: Devuelve todos los roles existentes en el sistema.
- **Actualizar rol (`PUT /api/roles/:id`)**: Permite modificar un rol existente.
- **Eliminar rol (`DELETE /api/roles/:id`)**: Permite eliminar un rol del sistema.

3. Permisos (Permission)

- **Crear permiso (`POST /api/permissions`)**: Permite crear nuevos permisos para el sistema.
- **Obtener permisos (`GET /api/permissions`)**: Devuelve todos los permisos disponibles.
- **Asignar permisos a roles (`POST /api/role-permissions/assign`)**: Permite asignar uno o varios permisos a un rol.

---

5. Recuperación de Contraseña

Se ha implementado la funcionalidad de **recuperación de contraseña**, que incluye los siguientes pasos:

- **Solicitud de recuperación (`POST /api/users/forgot-password`)**: Se envía un token de recuperación por correo electrónico.
- **Restablecimiento de contraseña (`POST /api/users/reset-password/:token`)**: Se verifica el token enviado al correo y permite al usuario restablecer su contraseña.

Para esta funcionalidad se utiliza **nodemailer** para enviar los correos y **bcryptjs** para la encriptación de contraseñas.

---

6. Seguridad

- **Encriptación de contraseñas**: Utilizamos **bcryptjs** para encriptar las contraseñas antes de almacenarlas en la base de datos.
- **Autenticación segura**: JWT asegura que solo los usuarios autenticados puedan acceder a rutas protegidas.
- **Protección de rutas**: Las rutas críticas están protegidas por roles y permisos, lo que garantiza que solo los usuarios con los privilegios adecuados puedan acceder a ciertas funcionalidades.

---

7. Tablas en la Base de Datos

Las tablas principales de la base de datos son:

- `users`: Almacena los datos de los usuarios del sistema.
- `roles`: Almacena los roles definidos en el sistema.
- `permissions`: Almacena los permisos del sistema.
- `role_permissions`: Tabla intermedia que almacena la relación muchos a muchos entre roles y permisos.

Conclusión y Estado Actual

Hemos implementado las funcionalidades básicas y esenciales de usuarios, roles, y permisos en el backend, con un sistema robusto de autenticación y autorización. El sistema está listo para conectarse al frontend, donde podremos mostrar y gestionar estos datos de manera visual.

El siguiente paso es la **integración del frontend con React** para ir materializando todo lo que hemos desarrollado en el backend.