# Proyecto: Repositorio de residencias

## Introducción

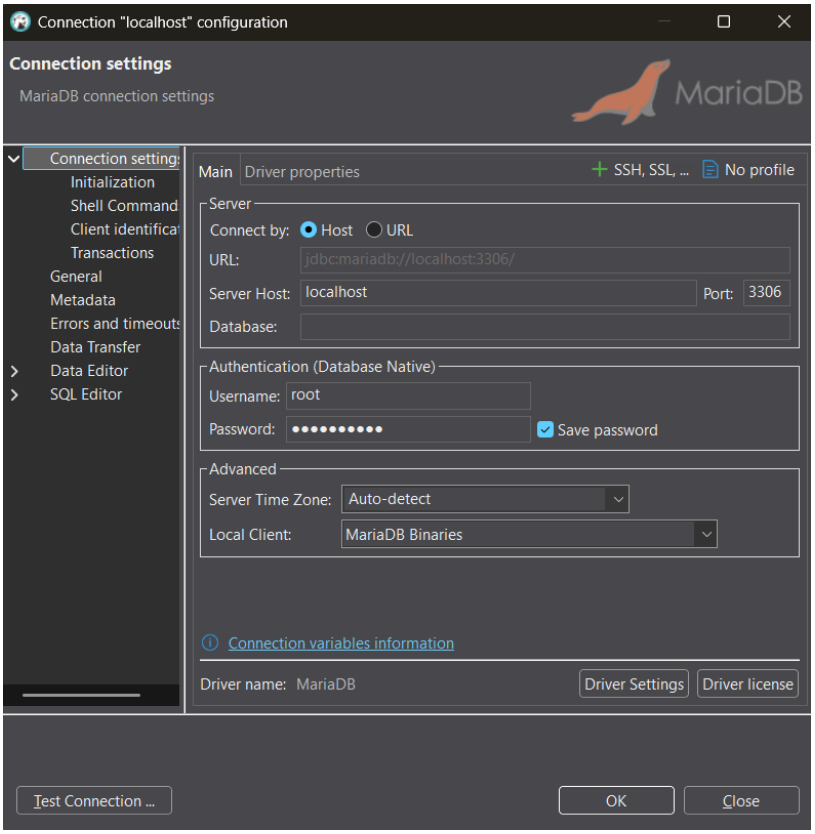
El propósito de este documento es facilitar el entendimiento y creación de un proyecto web, lo veremos desde la perspectiva de un fullstack iniciando con la creación de la base de datos (MariaDB), seguido de las APIRestful (Node + Fastify) y finalizando con el frontend (React), tomemos en cuenta que lo aprendido aquí no se limita a estas tecnológicas y puede ser aplicado utilizando SQL/Postgresql/Express/Vue/Angular etc... Analizaremos paso a paso por qué se hizo cada uno y el flujo de trabajo de la misma aplicación. Por ultimo este trabajo es escolar con un enfoque de trabajo estilo metricamovil. Cualquier duda me pueden preguntar ya saben. Los invito también a que hagan sus propios proyectos (Un gestor CRUD) con el esquema que se verá a continuación.

## Requisitos

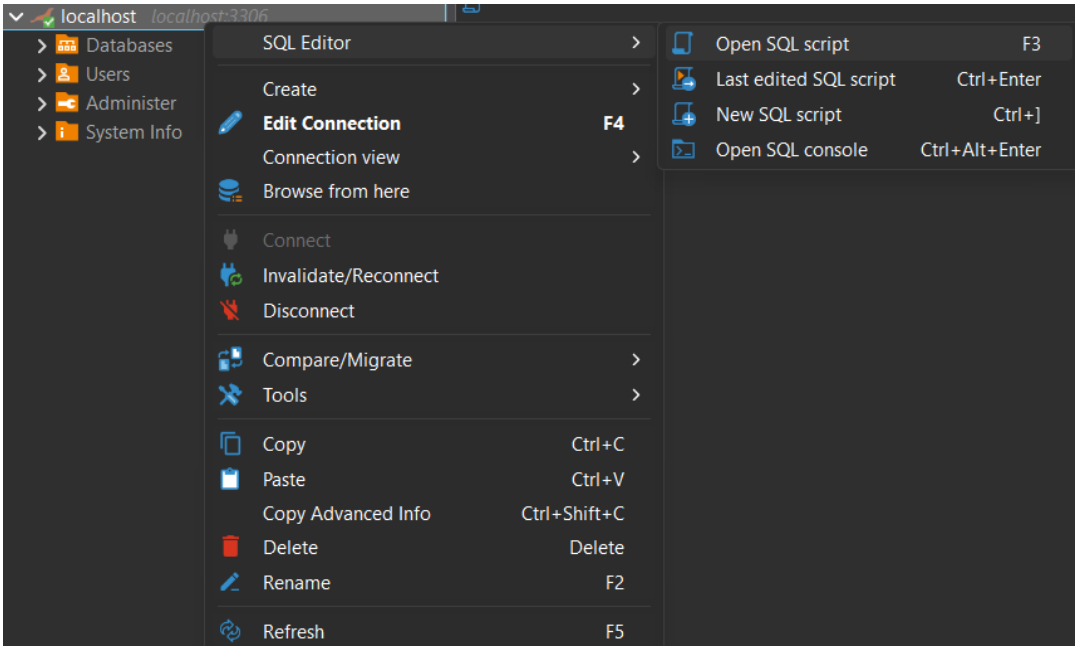
DBeaver con MariaDB
Visual Studio Code
Postman

## Base de Datos

Una vez instalado DBeaver pasamos a crear una conexión local utilizando MariaDB (Puede ser cualquier otro SQL)



Seguiremos con crear un nuevo script (también se puede realizar usando Ctrl + “]}`”)




Y lo que ya conocemos por las clases del Tec laguna, crear nuestra base de datos.

```sql
-- Creamos la base de datos
create database residencias;
-- Le decimos a la conexión que queremos usar esa (en caso de tener más de una)
use residencias;
```

En mi caso yo ya tengo definido qué tablas se utilizarán para este proyecto así que pondré a continuación los scripts realizados. *Dentro del script de SQL hay comentarios de porqué se hicieron algunas cosas por si hay dudas*

```sql
-- Por si se les llega a olvidar el use, se puede especificar también en qué database
-- queremos crear la tabla poniendo el nombre de la base de datos al comienzo de la tabla
-- decorandolo el nombre como se muestra a continuación
-- Creamos la tabla roles
create table residencias.roles (
    id bigint(20) unsigned not null auto_increment,
    role_name varchar(100) not null,
    description text default null,
    primary key (id),
    unique key uk_rolename (role_name)
) engine=InnoDB;
```

```sql
-- Creamos la tabla empresas
create table residencias.companies(
    id bigint(20) unsigned not null auto_increment,
    company_name varchar(255) not null,
    description text default null,
    address varchar(255) default null,
    phonenumber varchar(30) default null,
    email varchar(100) default null,
    primary key (id),
    unique key uk_name (company_name)
) engine=InnoDB;

-- Creamos la tabla modulos
-- Esta tabla es de las importantes ya que lo planeado para este proyecto
es que se tendrá
-- un sidebar para acceder a cada modulo, así podremos utilizar la access
key y definir
-- qué roles pueden acceder a qué y manejar mejor los permisos.
create table residencias.modules(
    id bigint(20) unsigned not null auto_increment,
    module_name varchar(20) not null,
    primary key (id),
    unique key uk_modulename (module_name)
) engine=InnoDB;

-- Creamos la tabla permisos
-- Otra de las tablas importantes aparte de asignar a qué módulos puede
entrar un usuario
-- con ese rol asignado, vamos a especificar qué permisos tendrá dentro
de ese módulo.
create table residencias.permissions(
    id bigint(20) unsigned not null auto_increment,
    permission varchar(50) not null,
    primary key (id),
    unique key uk_permname (permission)
) engine=InnoDB;

-- Creamos la tabla semestres
create table residencias.semester(
    id bigint(20) unsigned not null auto_increment,
    semester_name varchar(30) not null,
    primary key (id),
    unique key uk_sem_name (semester_name)
) engine=InnoDB;
```

```sql
-- Tabla de palabras clave (para buscar proyectos por tecnología
utilizada
create table residencias.keywords(
    id bigint(20) unsigned not null auto_increment,
    keyword varchar(30) default null,
    primary key (id),
    unique key uk_kw_name (keyword)
) engine=InnoDB;

-- Creamos nuestra tabla de usuarios (Ya que lo planeado es tener un
gestor
-- de usuarios.)
create table residencias.users(
    id bigint(20) unsigned not null auto_increment, -- Con estos
parametros estamos haciendo manualmente lo que hace el tipo 'serial',
podemos omitir todo eso y solamente poner serial y haría lo mismo.
    user_name varchar(100) not null, -- Necesitamos un user_name que sea
unique y not null
    password varchar(255) not null, -- Un usuario necesita si o si una
contraseña, porqué darle espacios se podrían preguntar y la respuesta es
porque vamos mas adelante a utilizar bcrypt para encriptarla y que sea
más dificil de hackear.
    email varchar(100) not null, -- No necesitamos tantos espacios ya que
se utilizará el correo institucional pero en caso de que no se use ese,
dejamos abierto a posibilidades.
    role_id bigint(20) unsigned not null, -- Al llegar a este punto nos
damos cuenta que en lo planeado es que el usuario tenga un rol, y este
rol será una foreign key de la tabla roles que aun no existe entonces
antes de crear esta tabla vamos a tener que hacer la tabla roles.
    is_active tinyint(1) default 1, -- este tipo de dato es lo
equivalente a tener un boleano, lo necesitamos para facilitarle al
administrador del proyecto la gestión de los usuarios.
    code varchar(10) default null, -- aquí es donde se va a almacenar el
código de recuperación lo dejamos por default null ya que puede o no
puede que se haya solicitado este código, esto lo veremos más adelante en
la sección de backend.
    primary key (id),
    unique key uk_user_name (user_name),
    key kid_role_id (role_id),
    constraint fk_user_roles foreign key (role_id) references
residencias.roles(id) on update cascade on delete cascade
) engine=InnoDB;
```

```sql
-- Probablemente la tabla más importante del proyecto, donde se
almacenará todo para poder presentarlo adecuadamente
create table residencias.reports(
    id bigint(20) unsigned not null auto_increment,
    student_name varchar(100) not null,
    ctrlnumber varchar(15) not null,
    major varchar(100) default null,
    report_title varchar(255) default null,
    company_id bigint(20) unsigned not null,
    pdf_route varchar(500) default null, -- La columna clave para una de
las funcionalidades del repositorio
    semester bigint(20) unsigned default null,
    primary key (id),
    unique key uk_ctrl_num (ctrlnumber),
    constraint fk_company_report foreign key (company_id) references
residencias.companies(id),
    constraint fk_sem_report foreign key (semester) references
residencias.semester(id)
) engine=InnoDB;

-- Ahora seguiremos con las tablas relacionales
-- Tabla roles x modulos
create table residencias.roles_modules(
    role_id bigint(20) unsigned not null,
    module_id bigint(20) unsigned not null,
    is_visible tinyint(1) default 1,
    primary key (role_id, module_id),
    key tk_module_id (module_id),
    constraint fk_roles_modulestr foreign key (role_id) references
residencias.roles(id),
    constraint fk_modules_rolestr foreign key (module_id) references
residencias.modules(id)
) engine=InnoDB;

-- Seguimos con permisos del rol para un módulo específico.
create table residencias.role_modules_permissions(
    role_id bigint(20) unsigned not null,
    module_id bigint(20) unsigned not null,
    permission_id bigint(20) unsigned not null,
    is_granted tinyint(1) default 1,
    key tk_module_roleper (module_id),
    key tk_perm_rolemod (permission_id),
    constraint fk_role_moduleperm foreign key (role_id) references
residencias.roles(id),
    constraint fk_module_roleperm foreign key (module_id) references
residencias.modules(id),
    constraint fk_perm_rolemodule foreign key (permission_id) references
residencias.permissions(id)
) engine=InnoDB;
```

```
-- Y nuestra ultima palabras clave x reportes, aquí he decidido separarlo
para tener un id por cada
-- palabra clave ejemplo 'JavaScript' y poder hacer búsquedas atravesó de
estas, se pudo también haber hecho
-- con un array de strings dentro de cada reporte y hacer la búsqueda
obteniendo los reportes que contengan
-- esa palabra, pero! cuando hagamos los procedures es muchísimo más
fácil hacerlo con dos tablas. Menos
-- trabajo de lógica de unir qué con que y joins y demás.
create table residencias.reports_keywords(
    report_id bigint(20) unsigned not null,
    keyword_id bigint(20) unsigned not null,
    key tk_rep_id (report_id),
    key tk_kword_id (keyword_id),
    constraint fk_reportid_kword foreign key (report_id) references
residencias.reports(id),
    constraint fk_kwordid_report foreign key (keyword_id) references
residencias.keywords(id)
) engine=InnoDB;
```

Ya terminamos con lo “sencillo” que es crear las tablas, ahora pasaremos a crear procedures para el manejo de la información CRUD, cómo ya mencioné es mucho más sencillo tener ya las funciones hechas y solo mandarlas llamar con un CALL dentro de Node, así nos evitamos la molestia de que nos equivoquemos en algún tipo de dato o sintaxis, solamente llamamos la función y mandamos los datos que pide y fin. Es una buena práctica aunque muchos prefieren usar algo como sequelize para comunicarse directamente con la BD, ambos son válidos y buenos ya depende del gusto de cada uno.

Recomiendo copiar y pegar todo en el script de DBeaver para que se más fácil de leer.

```
-- Procedures (procedimientos almacenados/funciones)
-- Aquí comenzaremos como fuimos haciendo las tablas
-- 1.- Roles HECHO, 2.- Empresas HECHO, 3.- Modulos HECHO, 4.- Permisos
HECHO, 5.- Semestre HECHO
-- 6.- Palabras clave HECHO, 7.- Usuarios HECHO, 8.- Reportes HECHO. Lo
mismo será con
-- las tablas relacionales HECHO.
-- Procedure create role
create or replace procedure residencias.create_role(
    in c_role_name varchar(100),
    in c_description text
)
begin
     -- validamos que el nombre que se va a insertar no esté vacio y sea
un nomnbre válido de caracteres
```

```
      if c_role_name is null or trim(c_role_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'Role name cannot be null or empty';
    end if;

      -- lo mismo con la descripción
      if c_description is null or trim(c_description) = ''
    then
        signal sqlstate '45000'
            set message_text = 'Description cannot be null or empty';
    end if;

    -- si ya existe un rol con ese nombre, lanzamos un error controlado
    if exists (
        select 1
        from residencias.roles
        where role_name = c_role_name
    ) then
        -- usamos signal sqlstate '45000' para generar un error
personalizado
        -- sqlstate '45000' es un código genérico para errores definidos
por el usuario
        signal sqlstate '45000'
            set message_text = 'Role name is already created';
    else
        -- si no existe, insertamos el nuevo rol
        insert into residencias.roles(role_name, description)
        values(c_role_name, c_description);
    end if;
end;

-- Procedure update role
create or replace procedure residencias.update_role(
    in u_role_id bigint,
    in u_new_role_name varchar(100),
    in u_new_description text
)
begin
      -- Aquí hacemos una verificación diferente en el create a mi me
gusta utilizar un simple
      -- if exists (select 1) ya que en teoria no debería de existir, se
podría hacer diferente
      -- se podría utilizar el nombre y sacar el id meterlo en una
variable y ahí ya ver si existe o no
      -- pero para un create es más rollo, donde si lo uso es en los
update y delete, recibo el id y si existe
      -- se actualiza o se borra, esto lo hacemos declarando una variable
puede tener el nombre que quieran
```

```
      -- pero yo uso v_exists para (verify if exists) o v_id (verify id)
son las que generalmente le pongo
      -- de nombre, he visto que algunos les ponen vId, _id, etc... ya
depende de cada uno

    declare v_exists tinyint unsigned; -- declaramos una varaible que
será tinyint porqué un tinyint?
    -- bueno pudimos haber usado también un bigint unsigned pero la razon
es que se utilizará para
    -- almacenar numeros 0-255 y se usarán como si fuera un boleano, =0
es que no existe, >0 existe
    -- el limite del tinyint es (0-255) y es más que suficiente para una
validación (para el tamaño de este proy.)

    if u_role_id is null then
        signal sqlstate '45000' set message_text = 'role id cannot be
null';
    end if;

    if u_new_role_name is null or trim(u_new_role_name) = '' then
        signal sqlstate '45000' set message_text = 'new role name cannot
be null or empty';
    end if;

    if u_new_description is null or trim(u_new_description) = '' then
        signal sqlstate '45000' set message_text = 'new description
cannot be null or empty';
    end if;

    -- Aquí es donde se utiliza haremos un count de todos los ids que
encontremos y ese numero se va
    -- a guardar en v_exists (sería muy parecido a select 1)
    -- verificamos existencia del rol por su id:
      -- hacemos count(*) para contar cuántos registros coinciden
(similar a usar exists o select 1)
      select count(*) into v_exists
        from residencias.roles
        where id = u_role_id;

      -- si no encontramos el rol
      if v_exists = 0 then
      -- lanzamos un error personalizado con signal y sqlstate '45000'
            signal sqlstate '45000'
                set message_text = 'role not found';
      else
    -- comprobamos que el nuevo nombre no esté asignado a otro rol
distinto
          select count(*) into v_exists
            from residencias.roles
            where role_name = u_new_role_name
```

```
       -- <> significa “diferente de”, para excluir el propio rol de la
comprobación
            and id <> u_role_id;

       -- si algún otro rol ya tiene ese nombre
          if v_exists > 0 then
       -- lanzamos un error indicando nombre duplicado
              signal sqlstate '45000'
                  set message_text = 'new role name already in use';
          end if;

         update residencias.roles
            set role_name  = u_new_role_name,
                description = u_new_description
          where id = u_role_id;
    end if;
end;


-- Procedure delete role
create or replace procedure residencias.delete_role(
      in d_role_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_role_id is null then
        signal sqlstate '45000'
            set message_text = 'role id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.roles
      where id = d_role_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'role not found';
    else
        delete from residencias.roles
         where id = d_role_id;
    end if;
end;


-- Procedure obtener todos los roles, aquí es importante decir que esta
nos serviría por ejemplo para un
-- combobox o para mostrar los datos en la tabla de roles,
create or replace procedure residencias.get_roles()
begin
```

```
    -- devolvemos todas las columnas que nos interesen de la tabla roles
    select
        id as role_id,
        role_name,
        description
    from residencias.roles;
end;

-- Aquí me desviaré un poco, ya que no necesitamos un CRUD de permisos (a
nivel interfaz usuario, pero
-- lo haré para manejar los permisos a nivel BD.)

-- Procedure obtener los permisos del rol para ese módulo
create or replace procedure residencias.get_permissions_by_rolemodule(
    in g_role_id int,
    in g_module_id int
)
begin
    if not exists (
        select 1
          from residencias.role_modules_permissions
          where role_id   = g_role_id
           and module_id = g_module_id
    ) then
        signal sqlstate '45000'
            set message_text = 'role or module is not found';
    else
        select
            rmp.permission_id,
            p.permission,
            rmp.is_granted
          from residencias.role_modules_permissions as rmp
          inner join residencias.permissions as p
            on rmp.permission_id = p.id
         where rmp.role_id = g_role_id
           and rmp.module_id = g_module_id;
    end if;
end;

-- Llegado a este punto creo que ya notamos que para obtener, asignar,
actualizar o borrar algo
-- primero mandamos los parametros de entrada (ya sea un id, o un nombre,
etc). El nombre que se
-- usa puede ser el que sea, podría ser inclusive get_permissions(pizza
varchar(100)) y funcionaría
-- yo personalmente prefiero usar la primer letra de lo que haré create =
c_, get = g_, delete = d_
-- muchos usan "name" como nombre de variable que es valido, entonces no
tienen que hacerlo igual
-- :)
```

```sql
-- Seguimos con las empresas

-- Procedure crear empresa
-- generalmente yo pongo hasta abajo del código un "/*" sin cerrar
-- ya que todo lo que esté de ahí hacía abajo no afecta lo demás y ahí
pego
-- los datos de la tabla para tenerlos durante la creacion de los
procedures
-- quedan algo así
-- `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
-- `company_name` varchar(255) NOT NULL,
-- `description` text DEFAULT NULL,
-- `address` varchar(255) DEFAULT NULL,
-- `phonenumber` varchar(30) DEFAULT NULL,
-- `email` varchar(100) DEFAULT NULL,
-- si necesitan pegar más obviamente sería lo ideal para que vayan viendo
qué datos
-- se van a manejar al crear el procedure (es un tip pero la mayoria creo
que lo hace así).
create or replace procedure residencias.create_company(
    in c_company_name varchar(255),
    in c_description text,
    in c_address varchar(255),
    in c_phonenumber varchar(30),
    in c_email varchar(100)
)
begin
      -- Aquí sólo por esta vez, les mostraré otra manera más
personalizada de hacer un mensaje
      -- de error, primero tenemos que declarar una variable tipo
varchar.
      declare v_msg varchar(400);

    if c_company_name    is null
       or trim(c_company_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'company name cannot be null or empty';
    end if;

    if c_description    is null
       or trim(c_description) = ''
    then
        signal sqlstate '45000'
            set message_text = 'description cannot be null or empty';
    end if;

    if c_address    is null
       or trim(c_address) = ''
```

```
    then
        signal sqlstate '45000'
            set message_text = 'address cannot be null or empty';
    end if;

    if c_phonenumber    is null
       or trim(c_phonenumber) = ''
    then
        signal sqlstate '45000'
            set message_text = 'phone number cannot be null or empty';
    end if;

    if c_email    is null
       or trim(c_email) = ''
    then
        signal sqlstate '45000'
            set message_text = 'email cannot be null or empty';
    end if;

    if exists (
        select 1
          from residencias.companies
         where company_name = c_company_name
    ) then
    -- Aquí hacemos un mensaje más personalizado esto nos serviría para
darle un toque más real,
    -- pero hacerlo así o no es lo mismo, sólo es avisar que hubo un
error de que ya existe.
    -- Este solo lo hice de ejemplo, seguiré haciendolo sin
    -- utilizarlo. ya que es mucho rollo tener que declarar variable y
demás
        set v_msg = concat('company "', c_company_name, '" is already
registered');
        signal sqlstate '45000'
            set message_text = v_msg;
    else
        -- si todo está bien, insertamos la nueva empresa
        insert into residencias.companies(company_name, description,
address, phonenumber, email)
        values(c_company_name, c_description, c_address, c_phonenumber,
c_email);
    end if;
end;

-- procedimiento para actualizar una empresa
create or replace procedure residencias.update_company(
    in u_company_id bigint,
    in u_new_name varchar(255),
    in u_new_description text,
    in u_new_address varchar(255),
```

```
    in u_new_phonenumber varchar(30),
    in u_new_email varchar(100)
)
begin
    declare v_exists tinyint unsigned;

    if u_company_id is null then
        signal sqlstate '45000'
            set message_text = 'company id cannot be null';
    end if;

    if u_new_name is null
       or trim(u_new_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new company name cannot be null or
empty';
    end if;
    if u_new_description is null
       or trim(u_new_description) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new description cannot be null or empty';
    end if;
    if u_new_address is null
       or trim(u_new_address) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new address cannot be null or empty';
    end if;
    if u_new_phonenumber is null
       or trim(u_new_phonenumber) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new phone number cannot be null or
empty';
    end if;
    if u_new_email is null
       or trim(u_new_email) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new email cannot be null or empty';
    end if;

    -- comprobamos existencia de la empresa
    select count(*) into v_exists
      from residencias.companies
      where id = u_company_id;

    if v_exists = 0 then
```

```
        signal sqlstate '45000'
            set message_text = 'company not found';
    else
        update residencias.companies
           set company_name = u_new_name,
               description  = u_new_description,
               address = u_new_address,
               phonenumber = u_new_phonenumber,
               email = u_new_email
         where id = u_company_id;
    end if;
end;

-- procedimiento para eliminar una empresa
create or replace procedure residencias.delete_company(
    in d_company_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_company_id is null then
        signal sqlstate '45000'
            set message_text = 'company id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.companies
      where id = d_company_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'company not found';
    else
        delete from residencias.companies
         where id = d_company_id;
    end if;
end;

-- procedure para obtener todas las empresas
create or replace procedure residencias.get_companies()
begin
    select
        id as company_id,
        company_name,
        description,
        address,
        phonenumber,
        email
    from residencias.companies;
end;
```

```sql
-- create_module
create or replace procedure residencias.create_module(
    in c_module_name varchar(20)
)
begin
    if c_module_name    is null
       or trim(c_module_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'module name cannot be null or empty';
    end if;

    if exists (
        select 1
          from residencias.modules
         where module_name = c_module_name
    ) then
        signal sqlstate '45000'
            set message_text = 'module name is already registered';
    else
        insert into residencias.modules(module_name)
        values(c_module_name);
    end if;
end;

-- get_modules
create or replace procedure residencias.get_modules()
begin
    select
        id as module_id,
        module_name
    from residencias.modules;
end;

-- actualizar modulo
create or replace procedure residencias.update_module(
    in u_module_id bigint,
    in u_new_name varchar(20)
)
begin
  declare v_exists tinyint unsigned;

  if u_module_id is null then
    signal sqlstate '45000' set message_text = 'module id cannot be
null';
  end if;

  if u_new_name is null or trim(u_new_name) = '' then
```

```
    signal sqlstate '45000' set message_text = 'new module name cannot be
empty';
  end if;

  select count(*) into v_exists
    from residencias.modules
   where id = u_module_id;

  if v_exists = 0 then
    signal sqlstate '45000' set message_text = 'module not found';
  else
    update residencias.modules
       set module_name = u_new_name
     where id = u_module_id;
  end if;
end;

-- Borrar modulo
create or replace procedure residencias.delete_module(
    in d_module_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_module_id is null then
        signal sqlstate '45000'
            set message_text = 'module id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.modules
     where id = d_module_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'module not found';
    else
        delete from residencias.modules
         where id = d_module_id;
    end if;
end;

-- crear permiso
create or replace procedure residencias.create_permission(
    in c_permission varchar(50)
)
begin

    if c_permission is null
       or trim(c_permission) = ''
```

```
    then
        signal sqlstate '45000'
            set message_text = 'permission cannot be null or empty';
    end if;

    if exists (
        select 1
          from residencias.permissions
         where permission = c_permission
    ) then
        signal sqlstate '45000'
            set message_text = 'permission already exists';
    else
        insert into residencias.permissions(permission)
        values(c_permission);
    end if;
end;

-- obtener permisos
create or replace procedure residencias.get_permissions()
begin
    select
        id as permission_id,
        permission
      from residencias.permissions;
end;

-- actualizar permiso
create or replace procedure residencias.update_permission(
    in u_permission_id bigint,
    in u_new_permission varchar(50)
)
begin
    declare v_exists tinyint unsigned;

    if u_permission_id is null then
        signal sqlstate '45000'
            set message_text = 'permission id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.permissions
      where id = u_permission_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'permission not found';
    end if;

    if u_new_permission is null
```

```
       or trim(u_new_permission) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new permission cannot be null or empty';
    end if;

    select count(*) into v_exists
      from residencias.permissions
     where permission = u_new_permission
       and id <> u_permission_id;
    if v_exists > 0 then
        signal sqlstate '45000'
            set message_text = 'permission name already in use';
    end if;

    update residencias.permissions
       set permission = u_new_permission
     where id = u_permission_id;
end;

-- borrar permiso
create or replace procedure residencias.delete_permission(
    in d_permission_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_permission_id is null then
        signal sqlstate '45000'
            set message_text = 'permission id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.permissions
     where id = d_permission_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'permission not found';
    end if;

    delete from residencias.permissions
     where id = d_permission_id;
end;

-- crear semestre
create or replace procedure residencias.create_semester(
    in c_semester_name varchar(30)
)
begin
```

```
    if c_semester_name is null
       or trim(c_semester_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'semester name cannot be null or empty';
    end if;

    if exists (
        select 1
          from residencias.semester
         where semester_name = c_semester_name
    ) then
        signal sqlstate '45000'
            set message_text = 'semester already exists';
    else
        insert into residencias.semester(semester_name)
        values(c_semester_name);
    end if;
end;

-- obtener todos los semestres
create or replace procedure residencias.get_semesters()
begin
    select
        id as semester_id,
        semester_name
      from residencias.semester;
end;

-- actualizar semester
create or replace procedure residencias.update_semester(
    in u_semester_id bigint,
    in u_new_name varchar(30)
)
begin
    declare v_exists tinyint unsigned;

    if u_semester_id is null then
        signal sqlstate '45000'
            set message_text = 'semester id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.semester
      where id = u_semester_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'semester not found';
    end if;
```

```
    if u_new_name is null
       or trim(u_new_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new semester name cannot be null or
empty';
    end if;

    select count(*) into v_exists
      from residencias.semester
     where semester_name = u_new_name
       and id <> u_semester_id;
    if v_exists > 0 then
        signal sqlstate '45000'
            set message_text = 'semester name already in use';
    end if;

    update residencias.semester
       set semester_name = u_new_name
     where id = u_semester_id;
end;

-- eliminar semestre
create or replace procedure residencias.delete_semester(
    in d_semester_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_semester_id is null then
        signal sqlstate '45000'
            set message_text = 'semester id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.semester
     where id = d_semester_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'semester not found';
    end if;

    delete from residencias.semester
     where id = d_semester_id;
end;

-- crear keyword
create or replace procedure residencias.create_keyword(
    in c_keyword varchar(30)
```

```
)
begin

      if c_keyword is null
       or trim(c_keyword) = ''
    then
        signal sqlstate '45000'
            set message_text = 'keyword cannot be null or empty';
    end if;

    if exists (
        select 1
          from residencias.keywords
         where keyword = c_keyword
    ) then
        signal sqlstate '45000'
            set message_text = 'keyword already exists';
    else
        insert into residencias.keywords(keyword)
        values(c_keyword);
    end if;
end;

-- obtener todas las keywords
create or replace procedure residencias.get_keywords()
begin
    select
        id as keyword_id,
        keyword
      from residencias.keywords;
end;

-- actualizar keyword
create or replace procedure residencias.update_keyword(
    in u_keyword_id bigint,
    in u_new_keyword varchar(30)
)
begin
    declare v_exists tinyint unsigned;

    if u_keyword_id is null then
        signal sqlstate '45000'
            set message_text = 'keyword id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.keywords
      where id = u_keyword_id;

    if v_exists = 0 then
```

```
        signal sqlstate '45000'
            set message_text = 'keyword not found';
    end if;

    if u_new_keyword is null
       or trim(u_new_keyword) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new keyword cannot be null or empty';
    end if;

    select count(*) into v_exists
      from residencias.keywords
     where keyword = u_new_keyword
       and id <> u_keyword_id;
    if v_exists > 0 then
        signal sqlstate '45000'
            set message_text = 'keyword already in use';
    end if;

    update residencias.keywords
       set keyword = u_new_keyword
     where id = u_keyword_id;
end;

-- eliminar keyword
create or replace procedure residencias.delete_keyword(
    in d_keyword_id bigint
)
begin
    declare v_exists tinyint unsigned;

    if d_keyword_id is null then
        signal sqlstate '45000'
            set message_text = 'keyword id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.keywords
     where id = d_keyword_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'keyword not found';
    end if;

    delete from residencias.keywords
     where id = d_keyword_id;
end;

-- crear usuario
```

```sql
create or replace procedure residencias.create_user(
    in c_user_name varchar(100),
    in c_password varchar(255),
    in c_email varchar(100),
    in c_role_id bigint
)
begin
    if c_user_name is null
       or trim(c_user_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'user name cannot be null or empty';
    end if;
    if c_password is null
       or trim(c_password) = ''
    then
        signal sqlstate '45000'
            set message_text = 'password cannot be null or empty';
    end if;
    if c_email is null
       or trim(c_email) = ''
    then
        signal sqlstate '45000'
            set message_text = 'email cannot be null or empty';
    end if;
    if c_role_id is null then
        signal sqlstate '45000'
            set message_text = 'role id cannot be null';
    end if;

    if exists (
        select 1
          from residencias.users
         where user_name = c_user_name
    ) then
        signal sqlstate '45000'
            set message_text = 'user name already exists';
    end if;

    if not exists (
        select 1
          from residencias.roles
         where id = c_role_id
    ) then
        signal sqlstate '45000'
            set message_text = 'assigned role not found';
    end if;

    insert into residencias.users(user_name, password, email, role_id)
    values(c_user_name, c_password, c_email, c_role_id);
```

```
end;

-- obtener todos los usuarios
create or replace procedure residencias.get_users()
begin
    select
        id as user_id,
        user_name,
        email,
        role_id,
        is_active,
        code
      from residencias.users;
end;

-- actualizar usuario
create or replace procedure residencias.update_user(
    in u_user_id bigint,
    in u_new_name varchar(100),
    in u_new_password varchar(255),
    in u_new_email varchar(100),
    in u_new_role_id bigint,
    in u_new_active tinyint
)
begin
    declare v_exists tinyint unsigned;

    if u_user_id is null then
        signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.users
      where id = u_user_id;

    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    if u_new_name is null
       or trim(u_new_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new user name cannot be null or empty';
    end if;
    if u_new_password is null
       or trim(u_new_password) = ''
    then
```

```
        signal sqlstate '45000'
            set message_text = 'new password cannot be null or empty';
    end if;
    if u_new_email is null
       or trim(u_new_email) = ''
    then
        signal sqlstate '45000'
            set message_text = 'new email cannot be null or empty';
    end if;
    if u_new_role_id is null then
        signal sqlstate '45000'
            set message_text = 'new role id cannot be null';
    end if;

    if u_new_active is null then
        signal sqlstate '45000'
            set message_text = 'active flag cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.users
     where user_name = u_new_name
       and id <> u_user_id;
    if v_exists > 0 then
        signal sqlstate '45000'
            set message_text = 'user name already in use';
    end if;

    select count(*) into v_exists
      from residencias.roles
     where id = u_new_role_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'assigned role not found';
    end if;

    update residencias.users
       set user_name = u_new_name,
           password = u_new_password,
           email = u_new_email,
           role_id = u_new_role_id,
           is_active = u_new_active
     where id = u_user_id;
end;

-- eliminar usuario
create or replace procedure residencias.delete_user(
    in d_user_id bigint
)
begin
```

```
    declare v_exists tinyint unsigned;

    if d_user_id is null then
        signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    select count(*) into v_exists
      from residencias.users
     where id = d_user_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    delete from residencias.users
     where id = d_user_id;
end;

-- crear reporte
create or replace procedure residencias.create_report(
    in c_student_name varchar(100),
    in c_ctrlnumber varchar(15),
    in c_major varchar(100),
    in c_report_title varchar(255),
    in c_company_id bigint unsigned,
    in c_pdf_route varchar(500),
    in c_semester bigint unsigned
)
begin

      if c_student_name is null
       or trim(c_student_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'student name cannot be null or empty';
    end if;
    if c_ctrlnumber is null
       or trim(c_ctrlnumber) = ''
    then
        signal sqlstate '45000'
            set message_text = 'control number cannot be null or empty';
    end if;

      if exists (
        select 1
          from residencias.reports
         where ctrlnumber = c_ctrlnumber
    ) then
        signal sqlstate '45000'
```

```
            set message_text = 'control number already exists';
    end if;

      if c_company_id is null then
        signal sqlstate '45000'
            set message_text = 'company id cannot be null';
    end if;
    if not exists (
        select 1
          from residencias.companies
         where id = c_company_id
    ) then
        signal sqlstate '45000'
            set message_text = 'company not found';
    end if;

      if c_semester is null then
        signal sqlstate '45000'
            set message_text = 'semester id cannot be null';
    end if;
    if not exists (
        select 1
          from residencias.semester
         where id = c_semester
    ) then
        signal sqlstate '45000'
            set message_text = 'semester not found';
    end if;

    insert into residencias.reports(student_name, ctrlnumber, major,
report_title, company_id, pdf_route, semester)
    values (c_student_name, c_ctrlnumber, c_major, c_report_title,
c_company_id, c_pdf_route, c_semester);
end;

-- obtener todos los reportes
create or replace procedure residencias.get_reports()
begin
    select
        id as report_id,
        student_name,
        ctrlnumber,
        major,
        report_title,
        company_id,
        pdf_route,
        semester as semester_id
      from residencias.reports;
end;
```

```sql
-- actualizar reporte
create or replace procedure residencias.update_report(
    in u_report_id bigint unsigned,
    in u_student_name varchar(100),
    in u_ctrlnumber varchar(15),
    in u_major varchar(100),
    in u_report_title varchar(255),
    in u_company_id bigint unsigned,
    in u_pdf_route varchar(500),
    in u_semester bigint unsigned
)
begin
    declare v_exists tinyint unsigned;

    if u_report_id is null then
        signal sqlstate '45000'
            set message_text = 'report id cannot be null';
    end if;
    select count(*) into v_exists
      from residencias.reports
     where id = u_report_id;
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'report not found';
    end if;

    if u_student_name is null
       or trim(u_student_name) = ''
    then
        signal sqlstate '45000'
            set message_text = 'student name cannot be null or empty';
    end if;

    if u_ctrlnumber is null
       or trim(u_ctrlnumber) = ''
    then
        signal sqlstate '45000'
            set message_text = 'control number cannot be null or empty';
    end if;

    select count(*) into v_exists
      from residencias.reports
     where ctrlnumber = u_ctrlnumber
       and id <> u_report_id;
    if v_exists > 0 then
        signal sqlstate '45000'
            set message_text = 'control number already in use';
    end if;

    if u_company_id is null then
```

```
        signal sqlstate '45000'
            set message_text = 'company id cannot be null';
    end if;
    if not exists (
        select 1
          from residencias.companies
         where id = u_company_id
    ) then
        signal sqlstate '45000'
            set message_text = 'company not found';
    end if;

    if u_semester is null then
        signal sqlstate '45000'
            set message_text = 'semester id cannot be null';
    end if;
    if not exists (
        select 1
          from residencias.semester
         where id = u_semester
    ) then
        signal sqlstate '45000'
            set message_text = 'semester not found';
    end if;

    update residencias.reports
       set student_name = u_student_name,
           ctrlnumber = u_ctrlnumber,
           major = u_major,
           report_title = u_report_title,
           company_id = u_company_id,
           pdf_route = u_pdf_route,
           semester = u_semester
     where id = u_report_id;
end;

-- eliminar reporte
create or replace procedure residencias.delete_report(
    in d_report_id bigint unsigned
)
begin
    declare v_exists tinyint unsigned;

    if d_report_id is null then
        signal sqlstate '45000'
            set message_text = 'report id cannot be null';
    end if;
    select count(*) into v_exists
      from residencias.reports
      where id = d_report_id;
```

```
    if v_exists = 0 then
        signal sqlstate '45000'
            set message_text = 'report not found';
    end if;

    delete from residencias.reports
     where id = d_report_id;
end;

-- Ya terminamos con el crud basico y el get permisos (que este es extra)
ahora faltó hacer unas funciones que
-- nos serán utiles por el tipo de interfaz y proyecto que se planea
hacer.
-- Asignar modulos al role
create or replace procedure residencias.assign_module_role(
    in a_role_id bigint unsigned,
    in a_module_id bigint unsigned,
    in a_is_visible tinyint
)
begin
    declare v_exists tinyint unsigned;

    if a_role_id is null then
        signal sqlstate '45000' set message_text = 'role id cannot be
null';
    end if;
    if a_module_id is null then
        signal sqlstate '45000' set message_text = 'module id cannot be
null';
    end if;

    select count(*) into v_exists from residencias.roles
     where id = a_role_id;
    if v_exists = 0 then
        signal sqlstate '45000' set message_text = 'role not found';
    end if;

    select count(*) into v_exists
     from residencias.modules
     where id = a_module_id;
    if v_exists = 0 then
        signal sqlstate '45000' set message_text = 'module not found';
    end if;

    insert into residencias.roles_modules(role_id, module_id, is_visible)
    values(a_role_id, a_module_id, a_is_visible)
    on duplicate key update
        is_visible = values(is_visible);
end;
```

```sql
-- asignar ppermisos al rol del módulo
create or replace procedure residencias.assign_permission_role_module(
    in a_role_id bigint unsigned,
    in a_module_id bigint unsigned,
    in a_permission_id bigint unsigned,
    in a_is_granted tinyint
)
begin
    declare v_exists tinyint unsigned;

    if a_role_id is null then
      signal sqlstate '45000'
            set message_text = 'role id cannot be null';
    end if;

    if a_module_id is null then
      signal sqlstate '45000'
            set message_text = 'module id cannot be null';
      end if;

    if a_permission_id is null then
      signal sqlstate '45000'
            set message_text = 'permission id cannot be null';
    end if;

    select count(*) into v_exists from residencias.roles
    where id = a_role_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'role not found';
    end if;

    select count(*) into v_exists from residencias.modules
    where id = a_module_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'module not found';
    end if;

    select count(*) into v_exists from residencias.permissions
     where id = a_permission_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'permission not found';
    end if;
```

```
    insert into residencias.role_modules_permissions(role_id, module_id,
permission_id, is_granted)
    values(a_role_id, a_module_id, a_permission_id, a_is_granted)
    on duplicate key update
    is_granted = values(is_granted);
end;

-- assign role to user
create or replace procedure residencias.assign_user_role(
    in a_user_id bigint unsigned,
    in a_role_id bigint unsigned
)
begin
    declare v_exists tinyint unsigned;

    if a_user_id is null then
      signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    if a_role_id is null then
      signal sqlstate '45000'
            set message_text = 'role id cannot be null';
    end if;

    select count(*) into v_exists from residencias.users
    where id = a_user_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    select count(*) into v_exists from residencias.roles
     where id = a_role_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'role not found';
    end if;

    update residencias.users
       set role_id = a_role_id
     where id = a_user_id;
end;

-- reset recovery code
create or replace procedure residencias.reset_recovery_code(
    in r_user_id bigint unsigned
)
```

```
begin
    declare v_exists tinyint unsigned;

    if r_user_id is null then
      signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    select count(*) into v_exists
    from residencias.users
    where id = r_user_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    update residencias.users
       set code = null
     where id = r_user_id;
end;

-- update user status
create or replace procedure residencias.update_user_status(
    in u_user_id   bigint unsigned,
    in u_is_active tinyint
)
begin
    declare v_exists tinyint unsigned;

    if u_user_id is null
      then signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    select count(*) into v_exists
    from residencias.users
    where id = u_user_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    update residencias.users
       set is_active = u_is_active
     where id = u_user_id;
end;

-- update user password (and clear code)
```

```sql
create or replace procedure residencias.update_user_password(
    in u_user_id      bigint unsigned,
    in u_new_password varchar(255)
)
begin
    declare v_exists tinyint unsigned;

    if u_user_id is null
      then signal sqlstate '45000'
                   set message_text = 'user id cannot be null';
    end if;

    select count(*) into v_exists
    from residencias.users
    where id = u_user_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    update residencias.users
       set password = u_new_password,
           is_active = 1,
           code = null
     where id = u_user_id;
end;

-- update basic user info
create or replace procedure residencias.update_user_info(
    in u_user_id bigint unsigned,
    in u_user_name varchar(100),
    in u_email varchar(100),
    in u_role_id bigint unsigned
)
begin
    declare v_exists tinyint unsigned;

    if u_user_id is null
      then signal sqlstate '45000'
            set message_text = 'user id cannot be null';
    end if;

    if u_user_name is null or trim(u_user_name) = '' then
      signal sqlstate '45000'
            set message_text = 'user name cannot be null or empty';
    end if;

    if u_email is null or trim(u_email) = '' then
      signal sqlstate '45000'
```

```
            set message_text = 'email cannot be null or empty';
    end if;

    if u_role_id is null then
      signal sqlstate '45000'
            set message_text = 'role id cannot be null';
    end if;

    select count(*) into v_exists
    from residencias.users
    where id = u_user_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'user not found';
    end if;

    select count(*) into v_exists
    from residencias.roles
    where id = u_role_id;

    if v_exists = 0 then
      signal sqlstate '45000'
            set message_text = 'role not found';
    end if;

    update residencias.users
       set user_name = u_user_name,
           email     = u_email,
           role_id   = u_role_id
     where id = u_user_id;
end;

-- get modules by role
create or replace procedure residencias.get_modules_by_role(
    in g_role_id bigint unsigned
)
begin
    if g_role_id is null then
        signal sqlstate '45000' set message_text = 'role id cannot be
null';
    end if;

    select
        m.id as module_id,
        m.module_name,
        rm.is_visible
      from residencias.modules m
      left join residencias.roles_modules rm
        on m.id = rm.module_id
```

```
       and rm.role_id = g_role_id;
end;

-- Yo suelo hacer un ejemplo de todos para irme guíando, estaría bonito
poder memorizar todo
-- y agilizar la creación y prueba de todo pero de momento, otra cosa es
que lo hago en un script
-- diferente para poder usar ctrl + F y buscarlo más rapido.

CALL residencias.create_role(
    "role_name"           -- varchar(100),
    "description"         -- text
);

CALL residencias.update_role(
    123                   -- bigint (u_role_id),
    "new_role_name"       -- varchar(100),
    "new_description"     -- text
);

CALL residencias.delete_role(
    123                   -- bigint (d_role_id)
);

CALL residencias.get_roles();

CALL residencias.get_permissions_by_rolemodule(
    1                     -- int (g_role_id),
    2                     -- int (g_module_id)
);

CALL residencias.create_company(
    "company_name"        -- varchar(255),
    "description"         -- text,
    "address"             -- varchar(255),
    "phonenumber"         -- varchar(30),
    "email"               -- varchar(100)
);

CALL residencias.update_company(
    123                   -- bigint (u_company_id),
    "new_name"            -- varchar(255),
    "new_description"     -- text,
    "new_address"         -- varchar(255),
    "new_phonenumber"     -- varchar(30),
    "new_email"           -- varchar(100)
);

CALL residencias.delete_company(
    123                   -- bigint (d_company_id)
```

```
);

CALL residencias.get_companies();

CALL residencias.create_module(
    "module_name"          -- varchar(20)
);

CALL residencias.get_modules();

CALL residencias.update_module(
    123                    -- bigint (u_module_id),
    "new_module_name"      -- varchar(20)
);

CALL residencias.delete_module(
    123                    -- bigint (d_module_id)
);

CALL residencias.create_permission(
    "permission"           -- varchar(50)
);

CALL residencias.get_permissions();

CALL residencias.update_permission(
    123                    -- bigint (u_permission_id),
    "new_permission"       -- varchar(50)
);

CALL residencias.delete_permission(
    123                    -- bigint (d_permission_id)
);

CALL residencias.create_semester(
    "semester_name"        -- varchar(30)
);

CALL residencias.get_semesters();

CALL residencias.update_semester(
    123                    -- bigint (u_semester_id),
    "new_semester_name"    -- varchar(30)
);

CALL residencias.delete_semester(
    123                    -- bigint (d_semester_id)
);

CALL residencias.create_keyword(
```

```
    "keyword"             -- varchar(30)
);

CALL residencias.get_keywords();

CALL residencias.update_keyword(
    123                   -- bigint (u_keyword_id),
    "new_keyword"         -- varchar(30)
);

CALL residencias.delete_keyword(
    123                   -- bigint (d_keyword_id)
);

CALL residencias.create_user(
    "user_name"           -- varchar(100),
    "password"            -- varchar(255),
    "email"               -- varchar(100),
    1                     -- bigint (c_role_id)
);

CALL residencias.get_users();

CALL residencias.update_user(
    123,                  -- bigint (u_user_id),
    "new_name"            -- varchar(100),
    "new_password"        -- varchar(255),
    "new_email"           -- varchar(100),
    2,                    -- bigint (u_new_role_id),
    1                     -- tinyint (u_new_active)
);

CALL residencias.delete_user(
    123                   -- bigint (d_user_id)
);

CALL residencias.create_report(
    "student_name"        -- varchar(100),
    "ctrlnumber"          -- varchar(15),
    "major"               -- varchar(100),
    "report_title"        -- varchar(255),
    1,                    -- bigint (c_company_id),
    "/path/to/file.pdf"   -- varchar(500),
    1                     -- bigint (c_semester)
);

CALL residencias.get_reports();

CALL residencias.update_report(
    123,                  -- bigint (u_report_id),
```

```
    "student_name"          -- varchar(100),
    "ctrlnumber"            -- varchar(15),
    "major"                 -- varchar(100),
    "report_title"          -- varchar(255),
    1,                      -- bigint (u_company_id),
    "/new/path.pdf"         -- varchar(500),
    2                       -- bigint (u_semester)
);

CALL residencias.delete_report(
    123                     -- bigint (d_report_id)
);

CALL residencias.assign_module_role(
    1,                      -- bigint (a_role_id),
    2,                      -- bigint (a_module_id),
    1                       -- tinyint (a_is_visible)
);

CALL residencias.assign_permission_role_module(
    1,                      -- bigint (a_role_id),
    2,                      -- bigint (a_module_id),
    3,                      -- bigint (a_permission_id),
    1                       -- tinyint (a_is_granted)
);

CALL residencias.assign_user_role(
    123,                    -- bigint (a_user_id),
    2                       -- bigint (a_role_id)
);

CALL residencias.reset_recovery_code(
    123                     -- bigint (r_user_id)
);

CALL residencias.update_user_status(
    123,                    -- bigint (u_user_id),
    0                       -- tinyint (u_is_active)
);

CALL residencias.update_user_password(
    123,                    -- bigint (u_user_id),
    "new_password"          -- varchar(255)
);

CALL residencias.update_user_info(
    123,                    -- bigint (u_user_id),
    "user_name"             -- varchar(100),
    "email"                 -- varchar(100),
    2                       -- bigint (u_role_id)
```

```
);

CALL residencias.get_modules_by_role(
    1                    -- bigint (g_role_id)
);
```

Bueno antes de decir que ya terminamos con BD, tenemos que hacer pruebas.

Iré haciendo las pruebas aquí y los errores que vaya encontrando los iremos arreglando y el porqué falló (si falla algo). No pasa nada si hay errores es normal al programar o hacer algo, es mejor que haya errores y repararlos a huir de los errores.

Por ejemplo ya encontré no un error pero si algo que se tiene que cambiar, las keywords tienen que ser minúscula entonces lo modificaremos para que así funcione

```
create or replace procedure residencias.create_keyword(
    in c_keyword varchar(30)
)
begin
    declare v_kw varchar(30);

    if c_keyword is null
       or c_keyword = ''
    then
        signal sqlstate '45000'
            set message_text = 'keyword cannot be null or empty';
    end if;

    set v_kw = lower(c_keyword);

    if exists (
        select 1
          from residencias.keywords
         where keyword = v_kw
    ) then
        signal sqlstate '45000'
            set message_text = 'keyword already exists';
    else
        insert into residencias.keywords(keyword)
        values(v_kw);
    end if;
end;
```

También noté que nos faltó una función que retorne los el rol asignalo al usuario, sus modulos y permisos. Entonces procedemos a hacerlo:

```sql
create or replace procedure residencias.get_permissions_by_user(
    in g_user_id bigint unsigned
)
begin
    declare v_exists tinyint unsigned;

    if g_user_id is null then
        signal sqlstate '45000' set message_text = 'user id cannot be
null';
    end if;

    select count(*) into v_exists
      from residencias.users
     where id = g_user_id;
    if v_exists = 0 then
        signal sqlstate '45000' set message_text = 'user not found';
    end if;

    -- seleccionamos:
    --   u.role_id                      -> id del rol asignado al usuario
    --   r.role_name                    -> nombre del rol
    --   m.id as module_id              -> id de cada módulo asociado al rol
    --   m.module_name                  -> nombre de cada módulo
    --   rm.is_visible                  -> visibilidad del módulo para ese rol (1 =
visible, 0 = oculto)
    --   p.id as permission_id          -> id de cada permiso asociado
    --   p.permission                   -> nombre del permiso
    --   ifnull(rmp.is_granted,0) as is_granted, esta función utiliza una expresión
    --   y un valor alternativo (en este caso elegimos el 0) por lo tanto
    --   revisa si el campo es 0 o 1 y si no existe ninguna fila para ese rol y mod
    --   al hacer el left join el valor será NULL (0) y lo pasa a 0 automatico.

      select
        u.role_id,
        r.role_name,
        m.id           as module_id,
        m.module_name,
        rm.is_visible,
        p.id           as permission_id,
        p.permission,
        ifnull(rmp.is_granted,0) as is_granted
      from residencias.users u
      join residencias.roles r
        on u.role_id = r.id
      join residencias.roles_modules rm
        on rm.role_id = r.id
      join residencias.modules m
        on rm.module_id = m.id
      left join residencias.role_modules_permissions rmp
        on rmp.role_id = r.id
       and rmp.module_id = m.id
      left join residencias.permissions p
        on p.id = rmp.permission_id;
end;
```

Y bueno de pruebas ya terminamos ya podemos pasar a lo que sigue y es hacer el backend, utilizaremos NodeJS con Fastify (Alternativa a Express, tc) al inicio del siguiente bloque explicaré un poco qué es fastify y porqué decidí utilizarlo también cuál será el flujo de trabajo y porqué se decidió que el backend funcione así.

---

## Backend

Empezaré por explicar qué es fastify y porqué lo usaremos, también con su instalación y la instalación de ciertas dependencias.

Primero qué es fastify? Bueno fastify es un framework web para NodeJS, que está diseñado para ofrecer mayor rendimiento al ecosistema del proyecto y eso es porque aporta varias cosas buenas, aunque express es el más popular y más maduro fastify tiene ciertas ventajas contra express.

Primero está optimizado para minimizar la carga de peticiones http y eso lo hace ser 2 o 3 veces más rápido que express utiliza PINO como sistema de logging de baja latencia.

Permite definir un JSON schema por ruta lo que acelera la validación de datos de entrada y serializacion de respuestas.

Y cada plugin de fastify crea su propio scope de rutas, decoraciones y hooks.

También tiene soporte native con http-2 y typescript.

Y procedo a explicar ciertos términos antes de seguir.

**¿Qué es el logging?**
Registrar en disco o en consola mensajes sobre lo que hace tu aplicación (errores, tiempos de respuesta, datos de depuración…).

**¿Por qué “baja latencia”?**
La latencia es el tiempo de respuesta. Un logger de baja latencia añade muy poco retraso a cada petición al escribir sus mensajes.

**PINO** (pronunciado “pee-no”) es la librería de logging que usa Fastify por defecto.

> Está optimizada para ser rápida: escribe en formato JSON muy compacto y usa técnicas asíncronas para no bloquear el ciclo de eventos de Node.js.
>
> Puedes configurar distintos niveles de logs (info, warn, error…) y salidas (consola, archivos, servicios externos).

**¿Qué es JSON Schema?**

Es un estándar para describir la forma que debe tener un objeto JSON:

Tipos de datos (string, number, array…)

Campos obligatorios vs. opcionales

Longitudes mínimas/máximas, formatos (email, uuid…), etc.

**¿Por qué Fastify usa JSON Schema?**

**Validación automática**: rechaza peticiones cuyo cuerpo (body), parámetros (params) o query no cumplan el esquema, antes de que tu código de negocio se ejecute.

**Serialización rápida**: convierte tus objetos JavaScript a JSON para la respuesta de manera optimizada.

**¿Qué es un plugin en Fastify?**

Un plugin es un módulo que "extiende" tu servidor con rutas, decoraciones o hooks (mecanismos que veremos más abajo). Se registra mediante fastify.register().

**Scope de plugin**
Cada plugin crea un **ámbito aislado** donde puedes:

- Registrar rutas (.get(), .post()…) que solo existan dentro de ese plugin.
- Decorar objetos (request, reply o el propio servidor) sin contaminar espacios globales.
- Añadir hooks que se ejecutan antes o después de cada petición en ese plugin.

Esto favorece la modularidad y evita colisiones: dos plugins pueden usar el mismo nombre de ruta o decoración sin interferir.

**Hooks: puntos de extensión del ciclo de la petición**

Los **hooks** son funciones que Fastify ejecuta en determinados momentos del ciclo de vida de una petición. Los principales son:

- onRequest: justo al recibir la petición, antes de parsear headers o body.
- preParsing: antes de parsear el cuerpo (por si quieres modificarlo bruto).
- preValidation: justo antes de la validación de esquemas.
- preHandler: antes de entrar al handler de la ruta (ideal para autenticación, autorización…).
- onSend: justo antes de enviar la respuesta al cliente.
- onResponse: después de que la respuesta ya salió (ideal para logging de métricas).

Cada hook puede ser registrado a nivel global (fastify.addHook) o dentro de un plugin/route. Sirven para insertar lógica transversal (autenticación, cache, métricas, etc.) sin ensuciar tus handlers.

---

## Backend 1.2 – Instalación de dependencias

Y bueno habiendo terminado con la breve explicación ahora si empezaremos con el proyecto, primero habrá que abrir VSC y crear una nueva carpeta, pueden llamarla como quieran yo la llame residencias

Una vez dentro pasaremos a abrir una nueva consola Ctrl + Ñ en mi caso y adentro escribiremos cmd (a algunos les causa error sin el cmd) y dentro procederemos a instalar lo siguiente:

```
npm install fastify                          # Núcleo de Fastify

npm install @fastify/cors                    # Middleware CORS

npm install @fastify/helmet                  # Protecciones HTTP (cabeceras seguras)

npm install @fastify/compress                # Compresión gzip/deflate

npm install @fastify/formbody                # Parseo de bodies tipo x-www-form-
urlencoded

npm install mariadb                          # Driver nativo de MariaDB
```

```
npm install fastify-mariadb                  # Plugin para MariaDB

npm install @easterneas/fastify-sequelize    # Plugin fastify sequelize

npm install sequelize sequelize-cli mariadb. # seq servidor con mariaDB

npm install joi                              # Instala la librería de JOI para los
                                             validators

npm install dotenv dotenv-expand             # Para gestionar el entorno de NodeJS

npm install –-save-dev nodemon               # Para actualizar el servidor con cada
                                             cambio
```

*importante* después de instalar nodemon hay que modificar el package.json y agregar

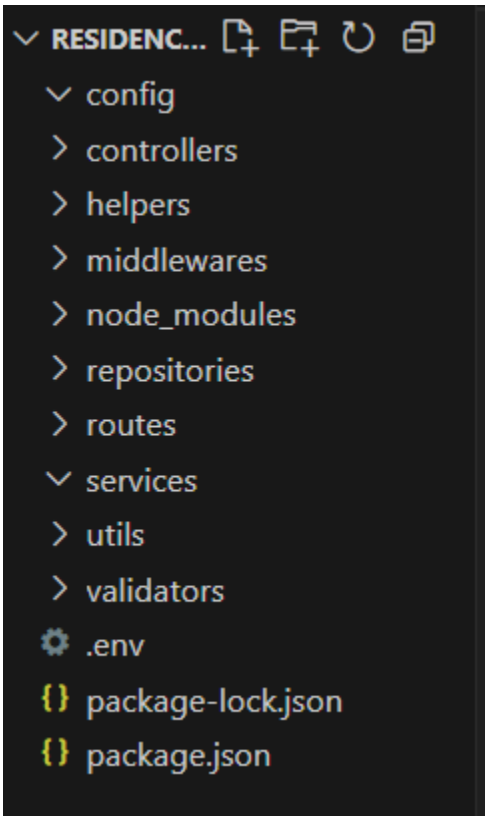
```
"scripts": {

  "dev": "nodemon src/index.js"

}
```

Les recomiendo que si package.json quede así:

```
{
  "type": "module",
  "name": "residencias",
  "version": "1.0.0",
  "description": "Proyecto Fastify con MariaDB, Sequelize y dotenv",
  "main": "config/server.js",
  "scripts": {
    "dev":   "nodemon config/server.js",
    "start": "node   config/server.js"
  },
  "dependencies": {
    "fastify": "^5.4.0",
    "fastify-mariadb": "^2.0.0",
    "@fastify/cors": "^11.0.1",
    "@fastify/helmet": "^13.0.1",
    "@fastify/compress": "^8.1.0",
    "@fastify/formbody": "^8.0.2",
    "mariadb": "^3.4.4",
    "sequelize": "^6.37.7",
    "sequelize-cli": "^6.6.3",
    "joi": "^17.13.3",
    "dotenv": "^17.0.1",
    "dotenv-expand": "^12.0.2"
  },
```

```
  "devDependencies": {
    "nodemon": "^3.1.10"
  },
  "author": "",
  "license": "MIT"
}
```

Teniendo las instalaciones y configuraciones entonces pasaremos a definir la estructura del proyecto y será la siguiente:




¿Qué irá en cada carpeta y porqué esas carpetas?

En **config** se centralizan todos los archivos de configuración de la aplicación:

[database.js]: Configuración de la conexión a la base de datos utilizando Sequelize.

[env.js]: Agrupa las variables de entorno en un solo objeto de configuración, haciendo el acceso a estas variables más ordenado.

[envSchema.js]: Valida que las variables de entorno tengan el tipo de dato correcto, lo que ayuda a evitar errores por configuración.

[server.js]: Configuración del servidor (puerto, middlewares globales, etc.).

**Controllers:** Gestionar la lógica de entrada y salida de datos.

[example.controller.js]: Recibe las peticiones, interactúa con los servicios y devuelve las respuestas adecuadas al cliente.

**Helpers:** Obtiene funciones secundarias o utilitarias que asisten el proceso de datos.

[example.helper.js]: Función de ayuda que puede ser reutilizada en distintos servicios o controladores.

**Middlewares:** Ejecuta procesos intermedios que se aplican antes de que las peticiones lleguen al controlador.

[example.middleware.js]: Middleware genérico para procesos comunes.

[auth.middleware.js]: Maneja la autenticación, ya sea por ApiKey o sessionId.

[error.middleware.js]: Captura y maneja errores que puedan ocurrir durante el flujo de la petición.

**Repositories:** Encapsula las llamadas y operaciones a la base de datos.

[example.repository.js]: Funciones específicas para interactuar con la base de datos, separando la lógica de acceso a datos del resto de la aplicación.

**Routes:** Define y agrupa las rutas de la API, contectando middlewares y controllers.

[example.routes.js]: Define las rutas específicas para un recurso y redirige a los correspondientes middlewares y controladores.

[health-check.routes.js]: Ruta para verificar que la API está en funcionamiento.

[index.routes.js]: Archivo que importa e integra todas las rutas de la aplicación.

**Services:** Contiene la lógica de negocio y procesamiento de datos.

[example.service.js]: Implementa las reglas y operaciones que necesita la aplicación para procesar datos y ejecutar la lógica central.

**Utils:** Son las funciones y utilidades reusables para todo el proyecto.

[example.util.js]: Fragmentos de código que pueden ser utilizados en distintos servicios o controladores, optimizando la reutilización y evitando duplicación.

**Validators:** Validan los datos de entrada, asegurando que cumplan con el formato y reglas definidas.

[example.validator.js]: Utiliza librerías como Joi para validar los datos que se reciben en las peticiones, garantizando que la API maneje datos correctos.

[.env]: Archivo donde se almacenan las variables de entorno de manera local. *Importante*: No se sube a GitHub para proteger información sensible.

## Backend 1.3 - Flujo de datos

[Petición desde Postman]: Se envía la solicitud HTTP.

[Routes]: La petición llega a una ruta específica definida en /routes.

[Middleware]: Se aplican uno o varios middlewares (por ejemplo, autenticación y manejo de errores) antes de procesar la petición.

[Validator]: Se valida que los datos entrantes cumplan con los esquemas definidos.

[Controller]: Se invoca el controlador correspondiente que gestiona la lógica de la petición.

[Service]: El controlador delega la lógica de negocio al servicio, que contiene el procesamiento central.

[Helper / Utils]: Se utilizan funciones auxiliares o utilidades para complementar el procesamiento.

[Repository]: Se realizan llamadas a la base de datos para guardar, modificar o recuperar información.

[Respuesta a Postman]: Finalmente, se envía la respuesta resultante de la operación.

## Backend 1.4 - Config

Comenzaremos con lo importate, tenemos que crear la conexión a la BD, una vez teniendo una conexión exitosa, ya podremos empezar lo demás. Entonces el primer archivo que hay que completar es .env.

Qué debe llevar .env? Como ya se mencionó son las variables de entorno y yo usaré las siguientes:

```
DATABASE_HOST=localhost
DATABASE_NAME=residencias
DATABASE_USER=root
DATABASE_PASSWORD=
DATABASE_PORT=3306
PORT=3000
```

Ahora dentro de config tenemos que definir nuestro esquema de dotenv. Creamos el archivo envSchema.js y llevará lo siguiente:

```
import dotenv from 'dotenv'
import dotenvexpand from 'dotenv-expand'
import Joi from 'joi'

const env = dotenv.config()
dotenvexpand.expand(env)

const envSchema = Joi.object({
    PORT: Joi.number().default(3000),
    DATABASE_HOST: Joi.string().required(),
    DATABASE_NAME: Joi.string().required(),
    DATABASE_USER: Joi.string().required(),
    DATABASE_PASSWORD: Joi.string().required(),
    DATABASE_PORT: Joi.number().default(3306),
    API_KEY: Joi.string().optional(),
}).unknown()

const { error, value: envValues} = envSchema.validate(process.env)

if (error) {
    console.error('Error de conexión: ', error.message)
    process.exit(1)
}

export {envValues}
```

Ahora explicaremos porqué se hizo lo que se hizo en envSchema

Los import dotenv y dotenv expand (llama las librerías que ya instalamos)

Comenzamos con const env = dotenv.config() aquí estamos definiendo que env lea lea el archivo .env y cargue cada KEY=VALUE en process.env

Dotenv expand es por si necesitamos utilizar otro tipo de sintaxis al mandar llamar las variables (Imaginemos que tenemos ciertos caracteres) un ejemplo sería si tuviéramos una URL donde haremos llamadas ejemplo

http://localhost:3000 con dotenvexpand sería "http://${HOST}:${PORT}"

Seguido de esto definimos el esquema JOI aquí estamos usando joi.object para describir las variables que esperamos de process.env. Basicamente validamos que los datos que nos mandaron sean los correctos y el tipo de dato sea el correcto. Definimos required porque sin este no puede funcionar la conexión a la bd y optional de que puede ser necesario o no, con default pasa lo mismo, siempre será ese el port que usaremos.

Al final validamos si hubo algún error, validate(process.env) comprueba que cada variable cumpla con su tipo.

Si hay algo incorrecto nos mandará un error.message explicando porqué falló.

Y después de encontrar un error detiene la aplicación con el code error: 1

Si todo está correcto mandamos hacer a envValues que es un objeto limpio, donde cada clave ya tiene el tipo correcto, los default aplicados y permite que se lean más fácil los datos, en vez de leer siempre proccess.env.PORT que siempre es string lee envValues.PORT y sabe que se trata de un número.

Porqué lo manejamos así?

Por seguridad, robustez: Evitamos un fallo crítico.

Autodocumentación: Muestra lo que esperamos de la aplicación.

Tipos correctos de datos: Pasa cada string crudo a números y boleanos donde debe de ir.

Configuración centralziada: Si se cambian variables o se añaden falta con cambiar este único esquema.

Errores claros: Nos dice exactamente donde y porqué falló agilizando el debugging.

Ya teniendo el envSchema pasaremos a hacer **env.js**

```
import { envValues } from './config/envSchema.js'

const envConfig = {
    database: {
        name: envValues.DATABASE_NAME,
        user: envValues.DATABASE_USER,
        password: envValues.DATABASE_PASSWORD,
        host: envValues.DATABASE_HOST,
        port: envValues.DATABASE_PORT,
    },
    global: {
```

```
        port: envValues.PORT,
    }
}

export { envConfig }
```

Primero importamos los valores ya validados en envSchema y agrupamos todo bajo database: aquí ya centralizamos la conexión con el servidor así cuando creamos nuestra instancia sequelize (o pool de mariadb) solo pasamos por envConfig.database

¿Por qué usar un objeto global? Aunque aquí sólo teganmos el port del servidor, en un futuro podríamos necesitar, por ejemplo, una bandera de "modo mantenimiento", la URL de un servicio externo, o cualquier otra variable de configuración general, esto aqune yo no lo usaré lo pongo para explicarlo. ¿Por qué leer envValues.PORT y no process.env.PORT? Así nos aseguramos de que es un número (no la cadena "3000") y de que, si no lo hemos definido, ya tiene el valor por defecto 3000.

Y exportamos nuestro un objeto envConfig con el cual se puede acceder a la base de datos o a .global.port, etc..

Ahora pasaremos a hacer nuestro archivo de conexión a la bd que será **database.js**:

```
import { Sequelize, QueryTypes } from 'sequelize'
import { envConfig } from './env.js'

const sequelize = new Sequelize(
  envConfig.database.name,
  envConfig.database.user,
  envConfig.database.password,
  {
    host: envConfig.database.host,
    port: envConfig.database.port,
    dialect: 'mariadb',
    logging: false,
    pool: {
      max: 10,
      min: 1,
      acquire: 30000,
      idle: 10000
    }
  }
)
```

```
const authenticate = async () => {
  try {
    await sequelize.authenticate()
    console.log('Conexión establecida con éxito.')
  } catch (error) {
    console.error('No se pudo conectar a la base de datos:', error.message)
    process.exit(1)
  }
}

const executeQuery = async (query, replacements = []) => {
  try {
    return await sequelize.query(query, {
      replacements,
      type: QueryTypes.SELECT
    })
  } catch (error) {
    throw new Error(`Error en la consulta a la BD: ${error.message}`)
  }
}

export { sequelize, authenticate, executeQuery }
```

Primero hacemos los import de sequelize (nuestra clase principal ORM que estaremos usando [ORM = Object Rational Mapping] para conectar a la bd y gestionar la bd.

Querytypes: indica el tipo de operación que haremos a la bd.

EnvConfig: nuestro objeto creado para la configuración de la conexión.

Creamos una instancia de sequelize la cual utiliza nuestros objetos anteriormente creados, desactivamos los logs de la consola de sql (podríamos usar los que hicimos nosotros en la bd pero esos solo son para gestionar las mismas calls y ver que todo estuviese bien al probar nuestras procedures.), y ahora vamos a configurar la pool de conexiones a la bd.

Lo que hacemos es que en

```
pool: {
      max: 10, -- Maximo de conexiones simultaneas posibles
      min: 1, -- Minimo de conexiones que se mantienen abiertas
      acquire: 30000, -- Tiempo máximo para adquirir una conexion
      idle: 10000 -- Tiempo tras el cual una conexión se libera
    }
```

Luego creamos la función authenticate, sequelize probará la conexión básica con la base de datos, si pasa nos dice que todo bien, si falla nos muestra el error y sale del proceso usando el código 1

ExecuteQuery ejecuta una función cruda (RAW SQL) en replacements es donde pondremos los parámetros para queries parametrizadas y en querytypes.SELECT hacemos literal un select en la BD, podría ser insert o el que querramos. También nos manda los errores capturados y encapsulados.

Al final hacemos un export de los tres modulos { sequelize, authenticate y executeQuery }

El servidor (server.js) lo haremos más adelante una vez que ya tengamos definidos algunos archivos.

---

Seguiremos ahora con definir unos archivos necesarios que utilziaremos en server.js

El primero no es necesario pero a mi me gusta usarlo, es en helpers-> un **catch.error.js:**

```
export async function catchError(promise) {
  try {
    const data = await promise;
    return [data, null];
  } catch (error) {
    return [null, error];
  }
}
```

Qué hace este código de bloque? Bueno es para no estar escribiendo a cada rato try catch, solamente eso, no recomiendo usarlo en controllers (aunque se puede) ya que ahí si necesitamos saber exactamente qué pasó igual se podría decorar para que funcione de mejor manera, pero donde yo lo uso es en los service.js

Y el segundo que tenemos que hacer es en middlewares -> un **error.middleware.js**:

```
function handleNotFound(request, reply) {
  const message = 'Ruta no encontrada'
  request.log.warn(message)

  reply.status(404).send({
    status: false,
    message,
    data: null,
  })
}
```

```
function handleError(error, request, reply) {
  const statusCode = error.statusCode || 500
  const message = error.message || 'Error en la petición'

  request.log.error(error)

  reply.status(statusCode).send({
    status: false,
    message,
    data: null,
  })
}

export { handleError, handleNotFound }
```

Este archivo lo que hace es, creamos una función handleNotFound y se ejecuta cada que hacemos una petición que no existe, es decir una ruta que escribimos mal o no tenemos definida nos dira´que hay error en la petición y nos manda un log de advertencia, la segunda parte el .status nos manda la respuesta de 404 NOT FOUND en nuestras llamadas usando las API y su cuerpo nos dice: status false (fallo lógico), message (ruta no encontrada) data: null (no hay datos que devolver) de este modo recibimos siempre un JSON consistente.

La segunda función handleError lo que hacemos es registrarla para hacer manejo de errores globales, se ejecuta cuando cualquier parte de nuestro código mande un error o rechaza una promesa sin manejar.

Usa error.statusCode sino está definido asume que es 500 INTERNAL SERVER ERROR.

Toma el mensaje error.message (este describe generalmente qué falló) y lo manda, nos sirve para debuggear, el cuerpo es el mismo al anterior y los mandaremos llamar más adelante en nuestro server.js

```
import Fastify from 'fastify'

// Ya cargamos y validamos .env en config/envSchema.js → env.js
import { envConfig } from '../config/env.js'
import { authenticate } from '../config/database.js'

// import router from '../routes/index.js'
import { handleError, handleNotFound } from
'../middlewares/error.middleware.js'

import fastifyCors    from '@fastify/cors'
import fastifyHelmet  from '@fastify/helmet'
import compress       from '@fastify/compress'
import fastifyFormbody from '@fastify/formbody'

const fastify = Fastify({
  logger: true,
  bodyLimit: 50 * 1024 * 1024,
})

// Middlewares de seguridad y parsing
fastify.register(fastifyHelmet, { contentSecurityPolicy: false })
fastify.register(fastifyCors,   { origin: true, optionsSuccessStatus: 200
})
fastify.register(fastifyFormbody)
fastify.register(compress)

// Decoraciones para tus middlewares personalizados

// Método helper para respuestas exitosas
fastify.decorateReply('sendSuccess', function({
  status     = true,
  statusCode = 200,
  message    = 'Operación exitosa',
  data       = null,
}) {
  this.status(statusCode).send({ status, message, data })
})

// Rutas bajo /api
// fastify.register(router, { prefix: '/api' })

// Handlers de error y ruta no encontrada
fastify.setErrorHandler(handleError)
fastify.setNotFoundHandler(handleNotFound)
```

```
// Ruta raíz de comprobación
fastify.get('/', async (request, reply) => {
  return reply.sendSuccess({
    message: 'API is running',
    data: {}
  })
})

const start = async () => {
  try {
    // 1) Verificar conexión a la BD
    await authenticate()
    fastify.log.info('Conexión a la base de datos establecida con
éxito.')

    // 2) Arrancar servidor en el puerto validado
    await fastify.listen({ port: envConfig.global.port })
    fastify.log.info(`Servidor corriendo en
http://localhost:${envConfig.global.port}`)
  } catch (err) {
    fastify.log.error(err)
    process.exit(1)
  }
}

start()
```

Empezamos a hacer el server.js y ¿qué se necesita? Primeramente los import. envConfig: objeto validado con las variables de entorno (puerto, credenciales, etc.).

authenticate: función que prueba la conexión a la BD (Sequelize).

router: módulo donde definimos y agrupamos todas las rutas de la API.

handleError y handleNotFound: son los middlewares globales para gestionar errores y rutas inexistentes.

Plugins de Fastify (cors, helmet, compress, formbody): añaden seguridad y parsing automático.

Seguido creamos una instancia fastify donde activamos el sistema de logging (PINO) y en bodylimit ponemos un limite al tamaño de peticiones con lo que pusimos ahí bodyLimit: 50 * 1024 * 1024, decimos que no exceda los 50mb

Ahora registramos nuestros plugings **fastify.register()** qué hace el plugging de register? Básicamente todo lo que registramos dentro del plugging (rutas, hooks, decorate, etc) queda encapsulado en su propio scope así evita que haya colisiones de nombres y hace mas fácil reusar ese bloque en otros proyectos. Cada plugging puede recibir sus propias opciones por ejemplo en los hooks podemos usar (onRequest, preHandler, etc como ya había mencionado antes) y las decoraciones (decorate y decorateReply) que registremos no afectarán el resto del servidor

*** Injerta el contenido de ese plugin (rutas, hooks, decoraciones) en tu servidor, en un scope aislado y con sus propias opciones.***

El **fastify.decorate()** añade una propiedad al método reply y básicamente es para evitar repetir la misma lógica de formateo de respuestas, todas las rutas usan el mismo método para devolver datos (JSON) y se pueden crear cuantos necesitemos o querramos. (sendSuccess, sendError, paginate, etc..)

**** Extiende el prototipo de reply para añadir métodos o propiedades que puedas usar directamente en tus handlers, favoreciendo la consistencia y la reutilización de lógica de respuesta.***

Ahora fastify.get('/', …): Registrará un endpoint HTTP GET /

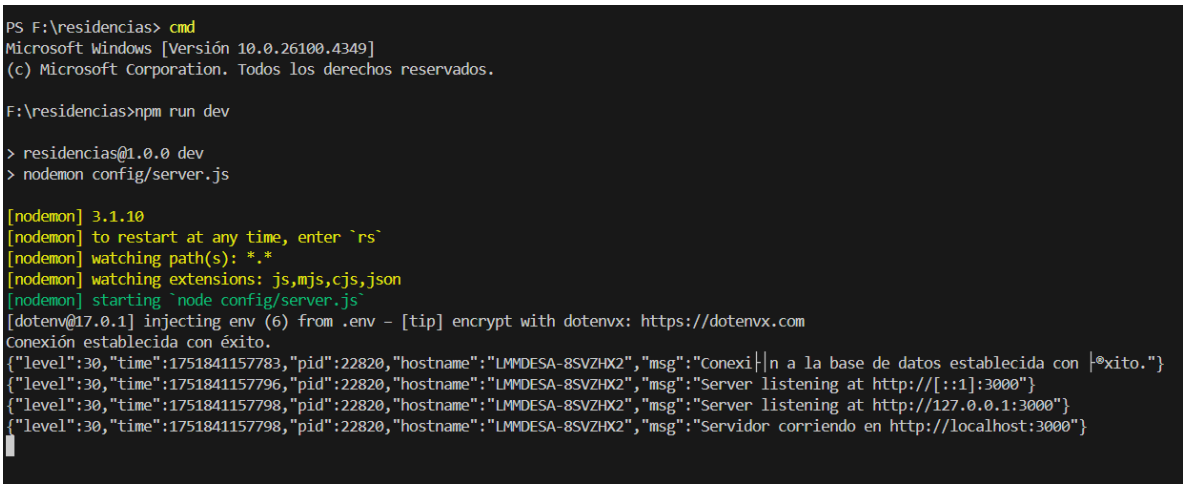
Y con reply.sendSuccess fija el código por defecto en 200 y envía un JSON con la estructura de status:true, message, data. El objetivo principal es que sirve como un "health check" si apuntamos el servidor usando postman con un localhost:3000 nos debería de devolver:

```
{
  "status": true,
  "message": "API is running",
  "data": {}
}
```

Nuestra función start manda llamar a al función que ya probó la conexión y si falla manda un error y cae al bloque catch, deteniendo el arranque.

El log se escriben ahí los logs que mandó la BD y arranca el servidor escuchando en el puerto definido anteriormente (envConfig.global.port) una vez levantado el sistema anota todo y maneja cuál quier error que pueda caer

Como nota. Aun no tenemos definido router (index.js) así que lo comentamos para que no cause error pero eventualmente lo vamos a utilizar, lo comenté para no borrarlo luego se me olvida ponerlo jaja igual iremos agregando más cosas al server.js pero de momento así funciona para que lo probemos, ya podemos hacer en la consola npm run dev y nos debería de arrojar un mensaje como el siguiente:




Nuestro servidor ya está corriendo sin problemas ☺ Ya podemos empezar a hacer todo lo que sigue del esquema de trabajo.

Teniendo en cuenta el flujo de datos del proyecto:

**Petición desde Postman**
El cliente (Postman, navegador, app móvil…) envía una solicitud HTTP al servidor, indicando método, ruta, headers y cuerpo.

**Routes**
Fastify busca la ruta correspondiente en /routes. Aquí sólo se enruta la petición al controlador o al siguiente middleware asociado a esa ruta.

**Middleware**
Antes de llegar al controlador:

***Autenticación/autorización:*** verifica tokens, sesiones o API-Keys.

***Logging, CORS, helmet…*** procesos transversales.

***Manejo de errores:*** captura excepciones tempranas.

**Validator**
Utiliza librerías como Joi para comprobar que request.body, params o query cumplen el esquema esperado. Si falla, se responde inmediatamente con un error de validación.

**Controller**
Aquí orquestamos la petición:

*Extraer datos de request.*

*Llamar al service adecuado.*

*Devolver la respuesta formateada (ejemplo, usando reply.sendSuccess()).*

**Service**
Contiene la **lógica de negocio pura**: reglas, cálculos, flujos de trabajo. No sabe de HTTP ni de BD, solo procesa datos.

**Helper / Utils**
Funciones auxiliares reutilizables (formateo de fechas, hashing de contraseñas, generación de códigos, etc.) que el service o controller pueden invocar.

**Repository**
Capa de acceso a datos: aquí usas Sequelize o consultas crudas para **persistir** o **recuperar** información en MariaDB. Devuelve entidades o resultados de consulta al service.

**Respuesta a Postman**

A mi me gustó empezar haciendo el repository ya que es la llamada directa a la función de la base de datos.

Debo decir antes de qué noté que olvidé retornar la id en las procedures de mariadb y por cuestiones de tiempo (ya que esto lo estoy haciendo en este fin de semana que acaba de pasar (5-6 jul) tendré que manejar diferente los repository, generalmente sólo haría un const rows = await executeQuery (‘residencias.create_role(?,?);’,

[role_name, description]

);

Return rows[0]; y así podemos saber si se creó exitosamente, pero como olvidé poner el return directo en la bd pues hay que adaptarlo diferente.

```
import { sequelize } from '../config/database.js'

class RolesRepository {
  async createRole({ role_name, description }) {
    const [, meta] = await sequelize.query(
      'CALL residencias.create_role(?, ?);',
      { replacements: [role_name, description] }
    )
    if (meta.affectedRows === 0) {
      throw new Error(`No se creó ningún rol (role_name=${role_name})`)
    }

    return { role_name, description }
  }

  async getRoles() {
    const [rows] = await sequelize.query(
```

```
      'CALL residencias.get_roles();'
    )
    return rows
  }

  async updateRole({ role_id, role_name, description }) {
    const [, meta] = await sequelize.query(
      'CALL residencias.update_role(?, ?, ?);',
      { replacements: [role_id, role_name, description] }
    )

    if (meta.affectedRows === 0) {
      throw new Error(`No se actualizó ningún rol (id=${role_id})`)
    }

    return { role_id, role_name, description }
  }

  async deleteRole({ role_id }) {
    const [, meta] = await sequelize.query(
      'CALL residencias.delete_role(?);',
      { replacements: [role_id] }
    )

    if (meta.affectedRows === 0) {
      throw new Error(`No se eliminó ningún rol (id=${role_id})`)
    }

    return { role_id }
  }
}

export default new RolesRepository()
```

¿Qué es lo que hacemos aquí?

Primero importamos el objeto sequelize que ya está configurado con la conexión y demás y a través de sequelize.query ejecutaremos los comandos de sql.

Definimos una clase RolesRepository y la exportamos hasta el final una instancia única donde vendrán los métodos que usaremos.

Empezamos con el create_role, ejecutamos el procedure (call residencias…) y lo hacemos con los parámetros de rolename y description, y aquí es donde

jugamos con lo que mencioné antes tenemos que saber que si se creó por lo tanto hacemos una validación con los últimos rows afectados de una llamada DML en sqlmariadb, sequelize.query devuelve un array [rows, meta], para un call de inserción, rows suele estar vacio meta.affectedRows indica cuantas filas se insertaron, si affectedRows termina siendo 0, se lanza un error que no se creó el rol. Devuelve un objeto {role_name, description} para que el service/controller que haremos ahorita sepan que se ha hecho una inserción.

---

```
import RolesRepository from '../repositories/roles.repository.js'
import { catchError } from '../helpers/catch.error.js'

class RolesService {
  async createRole(data) {
    const [result, error] = await
catchError(RolesRepository.createRole(data));
    if (error) throw error;
    return result;
  }

  async getRoles() {
    const [result, error] = await catchError(RolesRepository.getRoles());
    if (error) throw error;
    return result;
  }

  async updateRole(data) {
    const [result, error] = await
catchError(RolesRepository.updateRole(data));
    if (error) throw error;
    return result;
  }

  async deleteRole(data) {
    const [result, error] = await
catchError(RolesRepository.deleteRole(data));
    if (error) throw error;
    return result;
  }
}

export default new RolesService();
```

Esta parte de service es el intermedio que está entre el repository y los controllers (ahorita iremos a ello) básicamente lo que se hizo fue llamar a RolesRepository.metodo(data) y usar un catchError para capturar cualquier excepción, al usar catchError la capa nunca se rompe (UnhandledProiseRejection) y separa las responsabilidades del flujo (repository habla con sql y sus procedures, service es el manejo de errores y controller (que ahorita lo haremos) hace el request -> service -> response) y nos facilita c on las pruebas unitarias básicamente engloba el qué pasa si repository falla o no devuelve datos).

---

```
import RolesService from '../services/roles.service.js';

class RolesController {
  createRole = async (req, reply) => {
    const { role_name, description } = req.body
    const result = await RolesService.createRole({ role_name, description })
    reply.code(201).sendSuccess({ message: 'Role created', data: result })
  }

  getRoles = async (req, reply) => {
    const result = await RolesService.getRoles()
    reply.sendSuccess({ message: 'Roles fetched', data: result })
  }

  updateRole = async (req, reply) => {
    const { role_id, role_name, description } = req.body
    const result = await RolesService.updateRole({ role_id, role_name,
description })
    reply.sendSuccess({ message: 'Role updated', data: result })
  }

  deleteRole = async (req, reply) => {
    const { role_id } = req.params
    const result = await RolesService.deleteRole({ role_id })
    reply.code(204).sendSuccess({message: 'Role deleted', data: result})
  }
}


export default new RolesController();
```

El controlador, esta parte se encarga de encapsular los handlers de Fastify para gestionar operaciones CRUD sobre todos los recursos de roles, cada método recibe una petición **req** y un objeto de respuesta **reply**, llama al servicio correspondiente y devuelve una respuesta en formato estándar

(sendSuccess o status 200-204), solo explicaré como funciona el primero ya que los demás es igual, primero el método extrae role_name y description del cuerpo de la petición, llama a RolesService.createRole() y maneja la lógica de este, al obtener un resultado lo regresa así:

HTTP 201 Created

{

"status": true,

"message": "Role created",

"data": { "role_name": "...", "description": "..." }

}

Cualquier excepción lanzada por el servicio (por validaciones, fallos de BD, etc.) se propaga y es capturada por el error handler global (handleError) que formatea la respuesta con status: false y el mensaje de error correspondiente.

Por esta razón no envolvemos los controllers en try catch, dejamos que fastify lo maneje por si mismo.

---

Ya tenemos lo base para comenzar a trabajar y sólo nos faltan unos cuantos más antes de hacer las pruebas en postman, primero necesitaremos el validator, ¿qué es el validator? Como suena su nombre nos ayuda a validar, ¿qué valida? Que el dato ingresado sea el correcto para esa consulta.

```
import Joi from 'joi';

class RolesValidator {
    createRole() {
        return Joi.object({
            role_name: Joi.string().trim().required(),
            description: Joi.string().required(),
        });
    }

    getRoles(){
        return Joi.object({});
    }

    updateRoles(){
        return Joi.object({
            role_id: Joi.number().required(),
            role_name: Joi.string().trim().required(),
            description: Joi.string().required(),
        });
```

```
    }

    deleteRole(){
        return Joi.object({
            role_id: Joi.number().required(),
        });
    }
}

export default new RolesValidator();
```

No hay mucho que explicar, solamente utilizamos la librería Joi que nos sirve para valdiar de forma declarativa, definimos todo en un RolesValidator y lo exportamos

---

El validator lo necesitábamos para poder utilizar el middleware, ahora haremos roles.middleware.js:

```
import RolesValidator from "../validators/roles.validator.js";
import { handleError } from "./error.middleware.js";

class RolesMiddleware {
    createRole = async (req, reply) => {
        try {
            req.body = await
RolesValidator.createRole().validateAsync(req.body);
        } catch (err) {
            handleError(err, req, reply);
        }
    }

    getRoles = async (req, reply) => {
        try {
            req.query = await
RolesValidator.getRoles().validateAsync(req.query);
        } catch (err) {
            handleError(err, req, reply);
        }
    }

    updateRole = async (req, reply) => {
        try {
            req.body = await
RolesValidator.updateRole().validateAsync(req.body);
        } catch (err) {
```

```
            handleError(err, req, reply);
        }
    }

    deleteRole = async (req, reply) => {
        try {
            req.params = await
RolesValidator.deleteRole().validateAsync(req.params);
        } catch (err) {
            handleError(err, req, reply);
        }
    }
}

export default new RolesMiddleware();
```

Aquí podremos ver que si utilicé try catch a diferencia de los otros que utilicé el método de catchError ¿porqué? Porque mi helper catch.error está pensado para envolver promesas en repository o service donde quieres escribir un [resultado, error] y como aquí aparte utilizamos .validateAsync que lanza un ValidationError si falla el objetivo es interceptar ese error y manejarlo globalmente (handleError) para formatear la respuesta.

---

El siguiente archivo es roles.routes.js, este archivo es un fichero/grupo de endpoints HTTP que utiliza los controladores y los valida antes de ejecutarlos

```
import RolesController from '../controllers/roles.controller.js';

export default async function rolesRoutes(fastify, opts) {
    //POST /api/roles/create
    fastify.post(
        '/create', {preHandler: [fastify.rolesMiddleware.createRole]},
        RolesController.createRole
    );

    //GET /api/roles/list
    fastify.get(
        '/list', {preHandler: [fastify.rolesMiddleware.getRoles]},
        RolesController.getRoles
    );

    //PUT /api/roles/update
    fastify.put(
        '/update', {preHandler: [fastify.rolesMiddleware.updateRole]},
```

```
        RolesController.updateRole
    );

    //DELETE /api/roles/delete
    fastify.delete(
        '/delete/:role_id', {preHandler:
[fastify.rolesMiddleware.deleteRole]},
        RolesController.deleteRole
    );
}
```

Basicamente lo que hacen es extrae las instancias del import de Controller(que a su vez va conectado a todo lo demás), registramos la ruta como un plugin el cual tendrá un prefijo (este se da en server.js) es /api/roles, todo lo que se declare dentro de este bloque quedará bajo esa ruta base. El primero es POST, lo que hace primero usar el preHandler el cual ejecuta el middleware de validación (rolesMiddleware.createRole) que verifica el req.body contra les quema Joi (validator), si la validación pasa, entra en RolesController.createRole y crea el rol y respodne con el código 201 y así con los demás

```
import rolesRoutes from "./roles.routes.js";

async function router(fastify, opts) {
    fastify.register(rolesRoutes, {prefix: '/roles'})
}

export default router
```

Agregamos las rutas a nuestro index.js (ya que lo mandará llamar server.js)

```
Agregué el import necesario en server.js y
import RolesMiddleware from '../middlewares/roles.middleware.js'
```

Y el decorate:

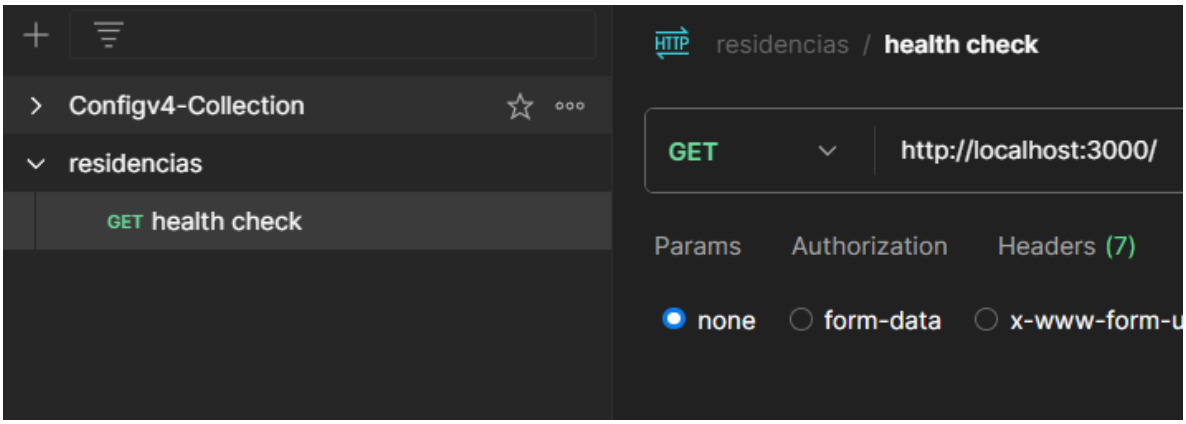
```
// Decoraciones para los middleware
fastify.decorate('rolesMiddleware', RolesMiddleware)
```

¿Qué hace? Añade una nueva propiedad fastify.rolesMiddleware cuyo valor es el objeto RolesMiddleware que importamos.

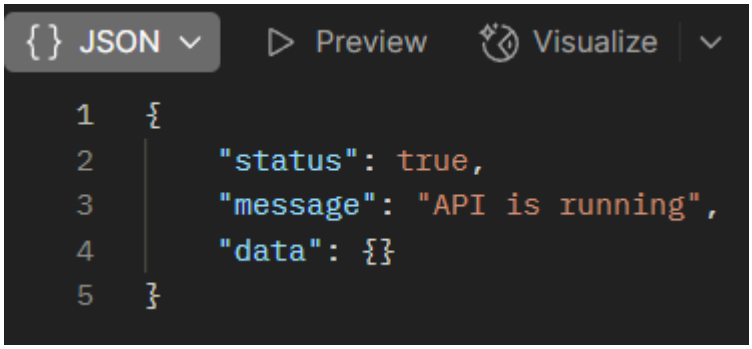
¿Para qué sirve? Centralizar nuestros middlewares personalizados en un único punto (normalmente en server.js).

Evitar tener que importar manualmente RolesMiddleware en cada fichero de rutas; en su lugar accederemos a él como fastify.rolesMiddleware dentro de cualquier plugin.

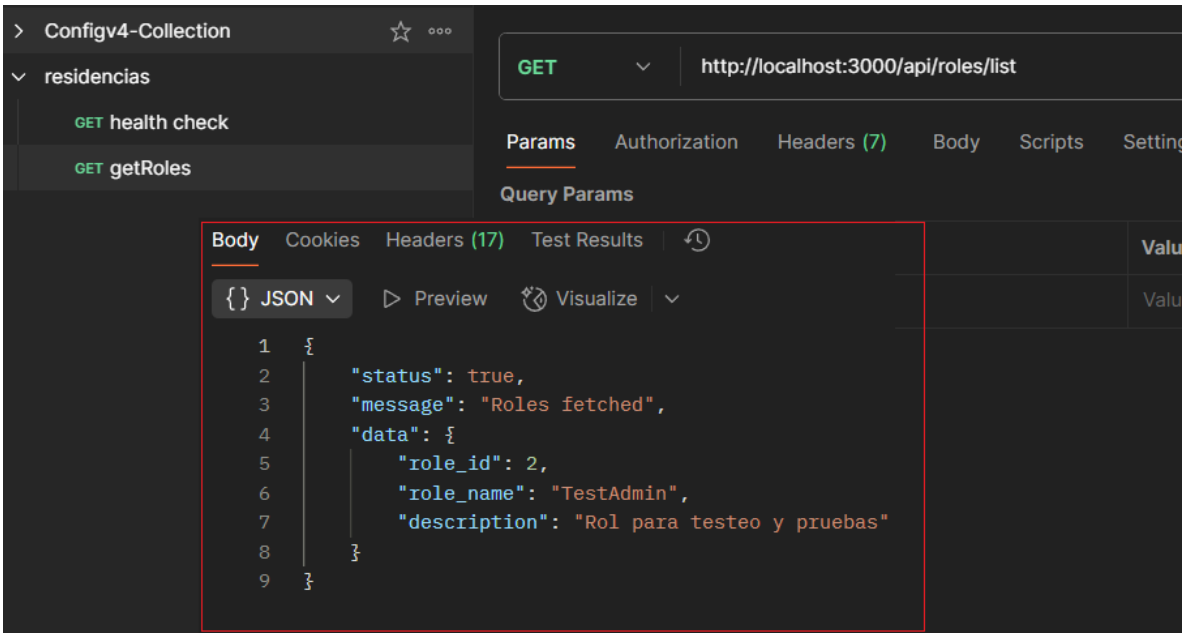
Y a continuación pasaremos a hacer las pruebas en Postman, simplemente creamos una nueva conexión y dentro de la conexión hacemos un neuvo request con lo siguiente:



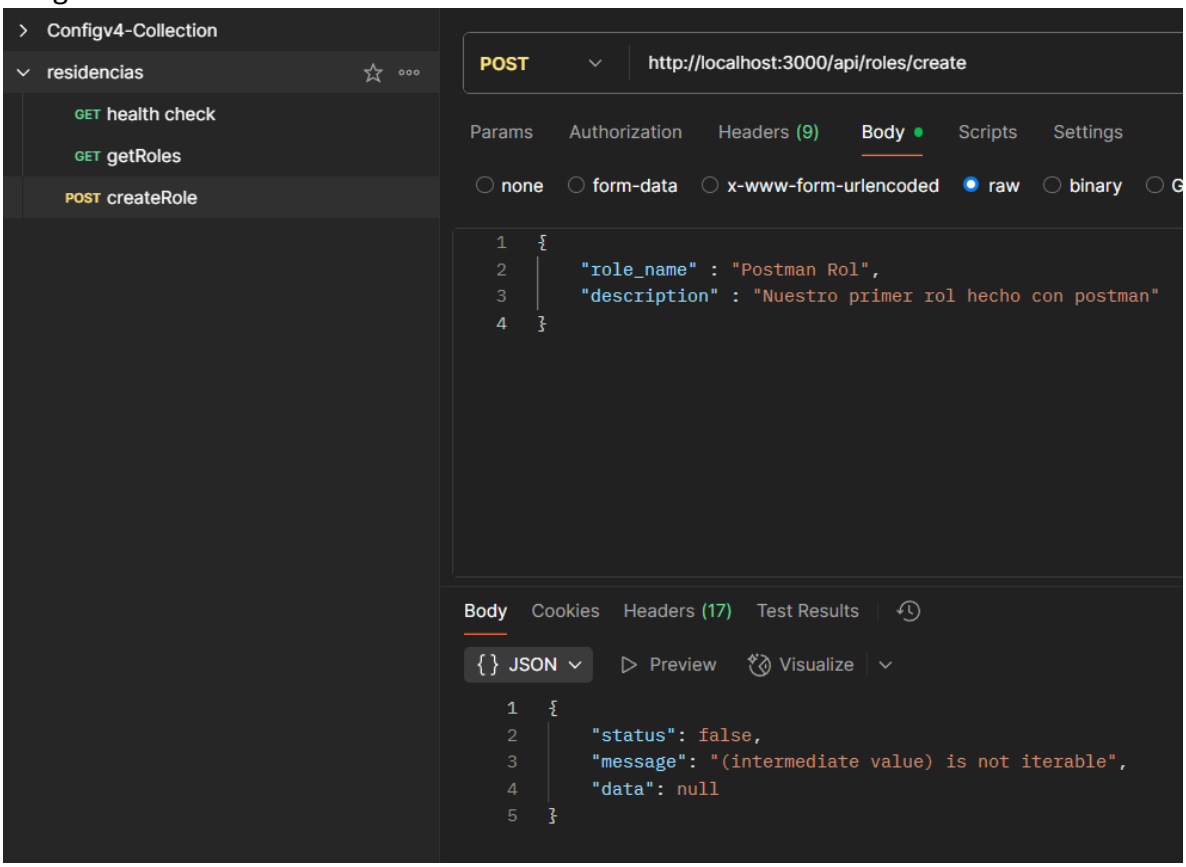

Este es para probar que el servidor está vivo, le damos send y nos devuelve:




Ahora probaremos las rutas que hicimos para roles, como yo ya hice de prueba un rol, lo mandaré llamar con el método de getRoles y debería de aparecer.




Como salió todo bien es una buena señal de que todo ha sido correctamente hecho, haré las demás pruebas para ver si todo bien, como es un POST para crear tenemos que cambiar el método y como elegimos hacer un req.body tenemos que escribirlo como si fuera un JSON como se muestra en la imagen:



Intenté crearlo pero me dio un error, entonces ahora hay que investigar qué significa ese error y porqué mandó error.

El error que nos manda en consola es:
:"::1","remotePort":65488},"msg":"incoming request"}

{"level":30,"time":1751854656675,"pid":11200,"hostname":"LMMDESA-8SVZHX2","reqId":"req-3","res":{"statusCode":200},"responseTime":9.481900006532669,"msg":"request completed"}

{"level":30,"time":1751854812037,"pid":11200,"hostname":"LMMDESA-8SVZHX2","reqId":"req-4","req":{"method":"POST","url":"/api/roles/create","host":"localhost:3000","remoteAddress":"::1","remotePort":49256},"msg":"incoming request"}

{"level":50,"time":1751854812052,"pid":11200,"hostname":"LMMDESA-8SVZHX2","reqId":"req-4",**"err":{"type":"TypeError","message":"(intermediate value) is not iterable","stack":"TypeError: (intermediate value) is not iterable\n at RolesRepository.createRole**

JavaScript espera un valor que regrese un sequelize.query(…), un array para poder destructurarlo con [, meta] pero tenemos error porque nosotros mandamos un objeto, entonces como investigué y en MariaDB no hay tal cosa, simplemente cambiarémos el código para que funcione

(file:///F:/residencias/repositories/roles.repository.js:5:22)\n    at process.processTicksAndRejections (node:internal/process/task_queues:105:5)\n    at async catchError (file:///F:/residencias/helpers/catch.error.js:3:18)\n    at async RolesService.createRole (file:///F:/residencias/services/roles.service.js:6:29)\n    at async Object.createRole (file:///F:/residencias/controllers/roles.controller.js:6:20)"},"msg":"(intermediate value) is not iterable"}

{"level":30,"time":1751854812053,"pid":11200,"hostname":"LMMDESA-8SVZHX2","reqId":"req-4","res":{"statusCode":500},"responseTime":15.627000004053116,"msg":"request completed"}

Esta es la línea responsable del error.

```
const [, meta] = await sequelize.query('CALL residencias.create_role(?, ?);',{ replacements: [role_name, description] })
```

Para solucionar el problema opté por omitir los arreglos y mejor dejé que la misma stored procedure se encargue si hay algún error, seguirá utilizando catch.error.js en caso de que algo salga mal y error.middleware.js entonces así evitaremos el uso de arrays, los únicos que si van a necesitar si o si son los get, ya que estos si devuelven datos (si hubiese hecho correcto las SP desde el comienzo, pudimos haber usado metadata para manejar el repository, ahí fue error mio pero no quiere decir que el proyecto ya esté mal, solo se trabajará diferente)

```
import { sequelize } from '../config/database.js';
import { sequelize } from '../config/database.js';
import { QueryTypes } from 'sequelize';

class RolesRepository {
  // Crea un nuevo rol. Si el SP falla, lanza excepción; si no, devolvemos
los datos pasados.
  async createRole({ role_name, description }) {
    await sequelize.query(
      'call residencias.create_role(?, ?);',
      { replacements: [role_name, description] }
    );
    return { role_name, description };
  }

  // Devuelve un array con todos los roles.
  async getRoles() {
    // usando QueryTypes.SELECT nos aseguramos de recibir directamente un
array de filas
    const rows = await sequelize.query(
      'call residencias.get_roles();',
      { type: QueryTypes.SELECT }
    );
    return rows;
  }

  async updateRole({ role_id, role_name, description }) {
    await sequelize.query(
      'call residencias.update_role(?, ?, ?);',
      { replacements: [role_id, role_name, description] }
    );
    return { role_id, role_name, description };
  }
```

```
  // Elimina un rol por su ID. El SP lanza error si no lo encuentra.
  async deleteRole({ role_id }) {
    await sequelize.query(
      'call residencias.delete_role(?);',
      { replacements: [role_id] }
    );
    return { role_id };
  }
}

export default new RolesRepository();
```

```
{
    "status": false,
    "message": "(conn:838, no: 1644, SQLState: 45000) Role name is
already created\nsql: CALL residencias.create_role('Postman Rol',
'Nuestro primer rol hecho con postman'); - parameters:[]",
    "data": null
}
```

Probando postman de nuevo con el POST, me di cuenta que a pesar del error anterior si creó el rol, entonces probaré delete para ver bien si está funcionando todo. Ya se arregló después de esos cambios y ya todo está bien, se ha arreglado el problema.

Ahora hay que darle nitro, ya que sabemos como se hacen los archivos del flujo de datos hay que hacer todos los SP que faltan.

Seguiré con empresas. A menos que salga un error ya no escribiré tanta explicación sólo si utilicé algo diferente pero no creo sea necesario.

---

Companies

```
import { sequelize } from '../config/database.js';
import { QueryTypes } from 'sequelize';

class CompaniesRepository {
    async createCompany({company_name, description, address, phonenumber,
email}){
        await sequelize.query(
            'call residencias.create_company(?,?,?,?,?);',
            { replacements: [company_name, description, address,
phonenumber, email]}
```

```
        );
        return {company_name, description, address, phonenumber, email};
    }

    async getCompanies(){
        const rows = await sequelize.query(
            'call residencias.get_companies();',
            { type: QueryTypes.SELECT }
        );
        return rows;
    }

    async updateCompany({company_id, company_name, description, address,
phonenumber, email}){
        await sequelize.query(
            'call residencias.update_company(?,?,?,?,?,?);',
            { replacements: [company_id, company_name, description, address,
phonenumber, email]}
        );
        return { company_id, company_name, description, address,
phonenumber, email};
    }

    async deleteCompany({company_id}){
        await sequelize.query(
            'call residencias.delete_company(?);',
            { replacements: [company_id]}
        );
        return {company_id};
    }

}

export default new CompaniesRepository();
```

```
import CompaniesRepository from '../repositories/companies.repository.js';
import { catchError } from '../helpers/catch.error.js';

class CompaniesService {
  async createCompany(data) {
    const [result, error] = await
catchError(CompaniesRepository.createCompany(data));
    if (error) throw error;
    return result;
  }
```

```
  async getCompanies() {
    const [result, error] = await
catchError(CompaniesRepository.getCompanies());
    if (error) throw error;
    return result;
  }

  async updateCompany(data) {
    const [result, error] = await
catchError(CompaniesRepository.updateCompany(data));
    if (error) throw error;
    return result;
  }

  async deleteCompany(data) {
    const [result, error] = await
catchError(CompaniesRepository.deleteCompany(data));
    if (error) throw error;
    return result;
  }
}

export default new CompaniesService();
```

```
import CompaniesService from '../services/companies.service.js';

class CompaniesController {
  createCompany = async (req, reply) => {
    const { company_name, description, address, phonenumber, email } =
req.body;
    const result = await CompaniesService.createCompany({ company_name,
description, address, phonenumber, email });
    reply.code(201).sendSuccess({ message: 'Company created', data: result
});
  }

  getCompanies = async (req, reply) => {
    const result = await CompaniesService.getCompanies();
    reply.sendSuccess({ message: 'Companies fetched', data: result });
  }

  updateCompany = async (req, reply) => {
```

```
    const { company_id, company_name, description, address, phonenumber,
email } = req.body;
    const result = await CompaniesService.updateCompany({ company_id,
company_name, description, address, phonenumber, email });
    reply.sendSuccess({ message: 'Company updated', data: result });
  }

  deleteCompany = async (req, reply) => {
    const { company_id } = req.params;
    const result = await CompaniesService.deleteCompany({ company_id });
    reply.code(204).sendSuccess({ message: 'Company deleted', data: result
});
  }
}

export default new CompaniesController();
```

```
import Joi from 'joi';

class CompaniesValidator {
  createCompany() {
    return Joi.object({
      company_name: Joi.string().trim().required(),
      description: Joi.string().required(),
      address: Joi.string().required(),
      phonenumber: Joi.string().required(),
      email: Joi.string().email().required(),
    });
  }

  getCompanies() {
    return Joi.object({});
  }

  updateCompany() {
    return Joi.object({
      company_id: Joi.number().required(),
      company_name: Joi.string().trim().required(),
      description: Joi.string().required(),
      address: Joi.string().required(),
      phonenumber: Joi.string().required(),
      email: Joi.string().email().required(),
    });
  }
```

```
  deleteCompany() {
    return Joi.object({
      company_id: Joi.number().required(),
    });
  }
}

export default new CompaniesValidator();
```

```
import CompaniesValidator from '../validators/companies.validator.js';
import { handleError } from './error.middleware.js';

class CompaniesMiddleware {
  createCompany = async (req, reply) => {
    try {
      req.body = await
CompaniesValidator.createCompany().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  }

  getCompanies = async (req, reply) => {
    try {
      req.query = await
CompaniesValidator.getCompanies().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  }

  updateCompany = async (req, reply) => {
    try {
      req.body = await
CompaniesValidator.updateCompany().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  }

  deleteCompany = async (req, reply) => {
    try {
```

```
      req.params = await
CompaniesValidator.deleteCompany().validateAsync(req.params);
    } catch (err) {
      handleError(err, req, reply);
    }
  }
}

export default new CompaniesMiddleware();
```

```
import CompaniesController from '../controllers/companies.controller.js';

export default async function companiesRoutes(fastify, opts) {
  // POST /api/companies/create
  fastify.post(
    '/create', { preHandler: [fastify.companiesMiddleware.createCompany]},
    CompaniesController.createCompany
  );

  // GET /api/companies/list
  fastify.get(
    '/list', { preHandler: [fastify.companiesMiddleware.getCompanies]},
    CompaniesController.getCompanies
  );

  // PUT /api/companies/update
  fastify.put(
    '/update', { preHandler: [fastify.companiesMiddleware.updateCompany]},
    CompaniesController.updateCompany
  );

  // DELETE /api/companies/delete/:company_id
  fastify.delete(
    '/delete/:company_id', { preHandler:
[fastify.companiesMiddleware.deleteCompany]},
    CompaniesController.deleteCompany
  );
}
```

Index.js: actualizado, el server.js lo mandaré al final.

```
import companiesRoutes from "./companies.routes.js"
import rolesRoutes from "./roles.routes.js";
```

```
async function router(fastify, opts) {
    fastify.register(rolesRoutes, {prefix: '/roles'})
    fastify.register(companiesRoutes, {prefix: '/companies'})
}

export default router
```

---

Modules

```
import { sequelize } from "../config/database";
import { QueryTypes } from "sequelize";

class ModulesRepository {
    async getModules(){
        const rows = await sequelize.query(
            'call residencias.getCompanies();',
            { type: QueryTypes.SELECT}
        );
        return rows;
    }

    async getModulesByRole({role_id}){
        await sequelize.query(
            'call residencias.get_modules_by_role(?);',
            { replacements: [role_id]}
        );
        return {role_id}
    }

    async assignModuleToRole({role_id, module_id, is_visible}){
        await sequelize.query(
            'call residencias.assign_module_role(?,?,?);',
            { replacements: [role_id, module_id, is_visible]}
        );
        return {role_id, module_id, is_visible}
    }
}

export default new ModulesRepository();
```

```
import ModulesRepository from '../repositories/modules.repository.js';
import { catchError } from '../helpers/catch.error.js';

class ModulesService {
```

```
  async getModules() {
    const [result, error] = await
catchError(ModulesRepository.getModules());
    if (error) throw error;
    return result;
  }

  async getModulesByRole(data) {
    const [result, error] = await
catchError(ModulesRepository.getModulesByRole(data));
    if (error) throw error;
    return result;
  }

  async assignModuleToRole(data) {
    const [result, error] = await
catchError(ModulesRepository.assignModuleToRole(data));
    if (error) throw error;
    return result;
  }
}

export default new ModulesService();
```

```
import ModulesService from '../services/modules.service.js';

class ModulesController {
  getModules = async (req, reply) => {
    const result = await ModulesService.getModules();
    reply.sendSuccess({ message: 'Modules fetched', data: result });
  };

  getModulesByRole = async (req, reply) => {
    const { role_id } = req.params;
    const result = await ModulesService.getModulesByRole({ role_id });
    reply.sendSuccess({ message: 'Modules by role fetched', data: result });
  };

  assignModuleToRole = async (req, reply) => {
    const { role_id, module_id, is_visible } = req.body;
    const result = await ModulesService.assignModuleToRole({ role_id,
module_id, is_visible });
    reply.code(201).sendSuccess({ message: 'Module assigned to role', data:
result });
  };
```

```
}

export default new ModulesController();
```

```
import Joi from 'joi';

class ModulesValidator {
  getModules() {
    return Joi.object({});
  }

  getModulesByRole() {
    return Joi.object({
      role_id: Joi.number().integer().required(),
    });
  }

  assignModuleToRole() {
    return Joi.object({
      role_id: Joi.number().integer().required(),
      module_id: Joi.number().integer().required(),
      is_visible: Joi.boolean().required(),
    });
  }
}

export default new ModulesValidator();
```

```
import ModulesValidator from '../validators/modules.validator.js';
import { handleError } from './error.middleware.js';

class ModulesMiddleware {
  getModules = async (req, reply) => {
    try {
      req.query = await
ModulesValidator.getModules().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  getModulesByRole = async (req, reply) => {
    try {
```

```
      req.params = await
ModulesValidator.getModulesByRole().validateAsync(req.params);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  assignModuleToRole = async (req, reply) => {
    try {
      req.body = await
ModulesValidator.assignModuleToRole().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };
}

export default new ModulesMiddleware();

--------------------------------------------------------------------------------

import ModulesController from '../controllers/modules.controller.js';

export default async function modulesRoutes(fastify, opts) {
  // GET  /api/modules/list
  fastify.get(
    '/list',
    { preHandler: [fastify.modulesMiddleware.getModules] },
    ModulesController.getModules
  );

  // GET  /api/modules/by-role/:role_id
  fastify.get(
    '/list-byrole/:role_id',
    { preHandler: [fastify.modulesMiddleware.getModulesByRole] },
    ModulesController.getModulesByRole
  );

  // POST /api/modules/assign
  fastify.post(
    '/assign',
    { preHandler: [fastify.modulesMiddleware.assignModuleToRole] },
    ModulesController.assignModuleToRole
  );
}
```

Permissions

```
import { sequelize } from "../config/database";
import { QueryTypes } from "sequelize";

class PermissionsRepository {
    async getPermissions() {
        const rows = await sequelize.query(
            'call residencias.get_permissions();',
            { type: QueryTypes.SELECT }
        );
        return rows;
    }

    async assignPermissions({role_id, module_id, permission_id,
is_granted}){
        await sequelize.query(
            'call residencias.assign_permission_role_module(?,?,?,?);',
            { replacements: [role_id, module_id, permission_id, is_granted]}
        );
    }

    async getPermissionsRoleModule ({role_id, module_id}){
        const rows = await sequelize.query(
            'call residencias.get_permissions_by_rolemodule(?,?);',
            {replacements: [role_id, module_id], type: QueryTypes.SELECT }
        );
        return rows;
    }

    async getPermissionsByUser ({user_id}){
        const rows = await sequelize.query(
            'call residencias.get_permissions_by_user(?);',
            {replacements: [user_id], type: QueryTypes.SELECT}
        );
        return rows;
    }

}

export default new PermissionsRepository();
```

```
import PermissionsRepository from
'../repositories/permissions.repository.js';
import { catchError } from '../helpers/catch.error.js';

class PermissionsService {
  async getPermissions() {
    const [result, error] = await
catchError(PermissionsRepository.getPermissions());
    if (error) throw error;
    return result;
  }

  async assignPermissions(data) {
    const [ result, error] = await
catchError(PermissionsRepository.assignPermissions(data));
    if (error) throw error;
    return result;
  }

  async getPermissionsRoleModule(data) {
    const [result, error] = await
catchError(PermissionsRepository.getPermissionsRoleModule(data));
    if (error) throw error;
    return result;
  }

  async getPermissionsByUser(data) {
    const [result, error] = await
catchError(PermissionsRepository.getPermissionsByUser(data));
    if (error) throw error;
    return result;
  }
}

export default new PermissionsService();

--------------------------------------------------------------------------

import PermissionsController from
'../controllers/permissions.controller.js';

export default async function permissionsRoutes(fastify, opts) {
  // GET  /api/permissions/list
  fastify.get(
    '/list',
```

```
    { preHandler: [fastify.permissionsMiddleware.getPermissions] },
    PermissionsController.getPermissions
  );

  // POST /api/permissions/assign
  fastify.post(
    '/assign',
    { preHandler: [fastify.permissionsMiddleware.assignPermissions] },
    PermissionsController.assignPermissions
  );

  // GET  /api/permissions/role-module/:role_id/:module_id
  fastify.get(
    '/list-rolemod/:role_id/:module_id',
    { preHandler: [fastify.permissionsMiddleware.getPermissionsRoleModule]
},
    PermissionsController.getPermissionsRoleModule
  );

  // GET  /api/permissions/user/:user_id
  fastify.get(
    '/list-permsuser/:user_id',
    {   compress: false,
        preHandler: [fastify.permissionsMiddleware.getPermissionsByUser] },
    PermissionsController.getPermissionsByUser
  );
}
```

---

Semesters

```
import { sequelize } from "../config/database.js";
import { QueryTypes } from "sequelize";

class SemestersRepository {
    async getSemesters(){
        const rows = await sequelize.query(
            'call residencias.get_semesters();',
            { type: QueryTypes.SELECT}
        );
        return rows;
    }
}
```

```
export default new SemestersRepository();
```

```
import SemestersRepository from '../repositories/semesters.repository.js';
import { catchError } from '../helpers/catch.error.js';

class SemestersService {
  async getSemesters() {
    const [result, error] = await catchError(
      SemestersRepository.getSemesters()
    );
    if (error) throw error;
    return result;
  }
}

export default new SemestersService();
```

```
import SemestersValidator from '../validators/semesters.validator.js';
import { handleError } from './error.middleware.js';

class SemestersMiddleware {
  getSemesters = async (req, reply) => {
    try {
      req.query = await
SemestersValidator.getSemesters().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };
}

export default new SemestersMiddleware();
```

```
import Joi from 'joi';

class SemestersValidator {
  getSemesters() {
    return Joi.object({});
  }
}

export default new SemestersValidator();
```

```
import SemestersValidator from '../validators/semesters.validator.js';
import { handleError } from './error.middleware.js';

class SemestersMiddleware {
  getSemesters = async (req, reply) => {
    try {
      req.query = await
SemestersValidator.getSemesters().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };
}

export default new SemestersMiddleware();
```

---

```
import SemestersController from '../controllers/semesters.controller.js';

export default async function semesterRoutes(fastify, opts) {
  // GET /api/semesters/list
  fastify.get(
    '/list',
    { preHandler: [fastify.semestersMiddleware.getSemesters] },
    SemestersController.getSemesters
  );
}
```

---

Keywords

```
import { sequelize } from '../config/database.js';
import { QueryTypes } from 'sequelize';

class KeywordsRepository {
    async createKeyword({keyword}){
```

```
        await sequelize.query(
            'call residencias.create_keyword(?);',
            { replacements: [keyword]}
        );
    return {keyword}
    }

    async getKeywords(){
        const rows = await sequelize.query(
            'call residencias.get_keywords();',
            {type: QueryTypes.SELECT}
        );
        return rows;
    }

    async deleteKeywords({keyword_id}){
        await sequelize.query(
            'call residencias.delete_keyword(?);',
            {replacements: [keyword_id]}
        );
        return {keyword_id}
    }
}

export default new KeywordsRepository();
```

```
import KeywordsRepository from '../repositories/keywords.repository.js';
import { catchError } from '../helpers/catch.error.js';

class KeywordsService {
  async createKeyword(data) {
    const [result, error] = await catchError(
      KeywordsRepository.createKeyword(data)
    );
    if (error) throw error;
    return result;
  }

  async getKeywords() {
    const [result, error] = await catchError(
      KeywordsRepository.getKeywords()
    );
    if (error) throw error;
    return result;
  }
```

```
  async deleteKeyword(data) {
    const [result, error] = await catchError(
      KeywordsRepository.deleteKeywords(data)
    );
    if (error) throw error;
    return result;
  }
}

export default new KeywordsService();
```

```
import KeywordsService from '../services/keywords.service.js';

class KeywordsController {
  createKeyword = async (req, reply) => {
    const { keyword } = req.body;
    const result = await KeywordsService.createKeyword({ keyword });
    reply.code(201).sendSuccess({ message: 'Keyword created', data: result
});
  };

  getKeywords = async (req, reply) => {
    const result = await KeywordsService.getKeywords();
    reply.sendSuccess({ message: 'Keywords fetched', data: result });
  };

  deleteKeyword = async (req, reply) => {
    const { keyword_id } = req.params;
    const result = await KeywordsService.deleteKeyword({ keyword_id });
    reply.code(204).sendSuccess({ message: 'Keyword deleted', data: result
});
  };
}

export default new KeywordsController();

import Joi from 'joi';

class KeywordsValidator {
  createKeyword() {
    return Joi.object({
      keyword: Joi.string().trim().required(),
    });
```

```
  }

  getKeywords() {
    return Joi.object({});
  }

  deleteKeyword() {
    return Joi.object({
      keyword_id: Joi.number().integer().required(),
    });
  }
}

export default new KeywordsValidator();
```

```
import KeywordsValidator from '../validators/keywords.validator.js';
import { handleError } from './error.middleware.js';

class KeywordsMiddleware {
  createKeyword = async (req, reply) => {
    try {
      req.body = await
KeywordsValidator.createKeyword().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  getKeywords = async (req, reply) => {
    try {
      req.query = await
KeywordsValidator.getKeywords().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  deleteKeyword = async (req, reply) => {
    try {
      req.params = await
KeywordsValidator.deleteKeyword().validateAsync(req.params);
    } catch (err) {
      handleError(err, req, reply);
    }
```

```
  };
}

export default new KeywordsMiddleware();

--------------------------------------------------------------------------------
import KeywordsController from '../controllers/keywords.controller.js';

export default async function keywordsRoutes(fastify, opts) {
  // POST /api/keywords/create
  fastify.post(
    '/create',
    { preHandler: [fastify.keywordsMiddleware.createKeyword] },
    KeywordsController.createKeyword
  );

  // GET /api/keywords/list
  fastify.get(
    '/list',
    { preHandler: [fastify.keywordsMiddleware.getKeywords] },
    KeywordsController.getKeywords
  );

  // DELETE /api/keywords/delete/:keyword_id
  fastify.delete(
    '/delete/:keyword_id',
    { preHandler: [fastify.keywordsMiddleware.deleteKeyword] },
    KeywordsController.deleteKeyword
  );
}
```

---

Users

```
import { QueryTypes } from 'sequelize';
import { sequelize } from '../config/database.js';

class UsersRepository {
  async createUser({ user_name, password, email, role_id }) {
    await sequelize.query(
      'call residencias.create_user(?,?,?,?);',
```

```
      { replacements: [user_name, password, email, role_id] }
    );
    return { user_name, password, email, role_id };
  }

  async getUsers() {
    const rows = await sequelize.query(
      'call residencias.get_users();',
      { type: QueryTypes.SELECT }
    );
    return rows;
  }

  async updateUser({ user_id, user_name, password, email, role_id, is_active
}) {
    await sequelize.query(
      'call residencias.update_user(?,?,?,?,?,?);',
      { replacements: [user_id, user_name, password, email, role_id,
is_active] }
    );
    return { user_id, user_name, password, email, role_id, is_active };
  }

  async deleteUser({ user_id }) {
    await sequelize.query(
      'call residencias.delete_user(?);',
      { replacements: [user_id] }
    );
    return { user_id };
  }
}

export default new UsersRepository();
```

```
import UsersRepository from '../repositories/users.repository.js';
import { catchError } from '../helpers/catch.error.js';

class UsersService {
  async createUser(data) {
    const [result, error] = await catchError(
      UsersRepository.createUser(data)
    );
    if (error) throw error;
    return result;
```

```
  }

  async getUsers() {
    const [result, error] = await catchError(
      UsersRepository.getUsers()
    );
    if (error) throw error;
    return result;
  }

  async updateUser(data) {
    const [result, error] = await catchError(
      UsersRepository.updateUser(data)
    );
    if (error) throw error;
    return result;
  }

  async deleteUser(data) {
    const [result, error] = await catchError(
      UsersRepository.deleteUser(data)
    );
    if (error) throw error;
    return result;
  }
}

export default new UsersService();
```

```
import UsersService from '../services/users.service.js';

class UsersController {
  createUser = async (req, reply) => {
    const { user_name, password, email, role_id } = req.body;
    const result = await UsersService.createUser({ user_name, password,
email, role_id });
    reply.code(201).sendSuccess({ message: 'User created', data: result });
  };

  getUsers = async (req, reply) => {
    const result = await UsersService.getUsers();
    reply.sendSuccess({ message: 'Users fetched', data: result });
  };
```

```
  updateUser = async (req, reply) => {
    const { user_id, user_name, password, email, role_id, is_active } =
req.body;
    const result = await UsersService.updateUser({ user_id, user_name,
password, email, role_id, is_active });
    reply.sendSuccess({ message: 'User updated', data: result });
  };

  deleteUser = async (req, reply) => {
    const { user_id } = req.params;
    const result = await UsersService.deleteUser({ user_id });
    reply.code(204).sendSuccess({ message: 'User deleted', data: result });
  };
}

export default new UsersController();
```

```
import Joi from 'joi';

class UsersValidator {
  createUser() {
    return Joi.object({
      user_name: Joi.string().trim().required(),
      password:  Joi.string().trim().required(),
      email:     Joi.string().email().required(),
      role_id:   Joi.number().integer().required(),
    });
  }

  getUsers() {
    return Joi.object({});
  }

  updateUser() {
    return Joi.object({
      user_id:   Joi.number().integer().required(),
      user_name: Joi.string().trim().required(),
      password:  Joi.string().trim().required(),
      email:     Joi.string().email().required(),
      role_id:   Joi.number().integer().required(),
      is_active: Joi.boolean().required(),
    });
  }
```

```
  deleteUser() {
    return Joi.object({
      user_id: Joi.number().integer().required(),
    });
  }
}

export default new UsersValidator();
```

```
import UsersValidator from '../validators/users.validator.js';
import { handleError } from './error.middleware.js';

class UsersMiddleware {
  createUser = async (req, reply) => {
    try {
      req.body = await UsersValidator.createUser().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  getUsers = async (req, reply) => {
    try {
      req.query = await UsersValidator.getUsers().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  updateUser = async (req, reply) => {
    try {
      req.body = await UsersValidator.updateUser().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  deleteUser = async (req, reply) => {
    try {
      req.params = await
UsersValidator.deleteUser().validateAsync(req.params);
    } catch (err) {
      handleError(err, req, reply);
    }
```

```
  };
}

export default new UsersMiddleware();
```

```
import UsersController from '../controllers/users.controller.js';

export default async function usersRoutes(fastify, opts) {
  // POST   /api/users/create
  fastify.post(
    '/create',
    { preHandler: [fastify.usersMiddleware.createUser] },
    UsersController.createUser
  );

  // GET    /api/users/list
  fastify.get(
    '/list',
    { preHandler: [fastify.usersMiddleware.getUsers] },
    UsersController.getUsers
  );

  // PUT    /api/users/update
  fastify.put(
    '/update',
    { preHandler: [fastify.usersMiddleware.updateUser] },
    UsersController.updateUser
  );

  // DELETE /api/users/delete/:user_id
  fastify.delete(
    '/delete/:user_id',
    { preHandler: [fastify.usersMiddleware.deleteUser] },
    UsersController.deleteUser
  );
}
```

---

Reports: Después de mucho código por fin viene algo diferente, aquí vamos a cambiar la manera de hacer varias cosas la razón es que necesitamos encontrar la manera de poder subir PDF’s a nuestro proyecto ya que los informes de residencia son en PDF. Por eso usaremos algo llamado Fastify-multipart (este puede recibir el pdf) también modificamos el SP de

créate_report y agregamos esta linea `select last_insert_id() as report_id;` para poder manejar como les había comentado antes si se realizó correctamente o no

```
import { sequelize } from '../config/database.js';
import { QueryTypes } from 'sequelize';

class ReportsRepository {
  async createReport({ student_name, ctrlnumber, major, report_title,
company_id, pdf_route, semester }) {
    const [rows] = await sequelize.query(
      'call residencias.create_report(?,?,?,?,?,?,?);',
      {
        replacements: [student_name, ctrlnumber, major, report_title,
company_id, pdf_route, semester],
        type: QueryTypes.SELECT
      }
    );
    return rows[0].report_id;
  }

  async getReports() {
    const rows = await sequelize.query(
      'call residencias.get_reports();',
      { type: QueryTypes.SELECT }
    );
    return rows;
  }

  async updateReport({ report_id, student_name, ctrlnumber, major,
report_title, company_id, pdf_route, semester }) {
    await sequelize.query(
      'call residencias.update_report(?,?,?,?,?,?,?,?);',
      {
        replacements: [report_id, student_name, ctrlnumber, major,
report_title, company_id, pdf_route, semester]
      }
    );
  }

  async deleteReport({ report_id }) {
    await sequelize.query(
      'call residencias.delete_report(?);',
      { replacements: [report_id] }
    );
  }
```

```
}

export default new ReportsRepository();
```

Ahora comienzo a explicar qué se tuvo que cambiar?, bueno, básicamente aquí todo se quedó igual que los demás con la excepción de que cambias el crear reporte para que nos devuelva la id del ultimo archivo insertado (con el fin de saber si se insertó bien o no).

```
import ReportsRepository from '../repositories/reports.repository.js';
import KeywordsRepository from '../repositories/keywords.repository.js';
import { catchError } from '../helpers/catch.error.js';
import { sequelize } from '../config/database.js';

class ReportsService {
  async createReport(data, file) {
    return await sequelize.transaction(async (tx) => {
      // guardar el PDF ya movido en controller como file.path
      const pdf_route = file.path.replace(/^.*reportes/, 'reportes');
      // crear reporte y obtener ID
      const [report_id, err1] = await catchError(
        ReportsRepository.createReport({ ...data, pdf_route }), tx
      );
      if (err1) throw err1;

      // asociar cada palabra clave
      for (const palabra of data.palabras_clave || []) {
        // obtener o crear palabra
        let [rows] = await tx.query(
          'select palabra_id from residencias.palabras_clave where palabra =
?',
          { replacements: [palabra], type: QueryTypes.SELECT }
        );
        let palabra_id = rows.length ? rows[0].palabra_id
          : (await tx.query(
              'insert into residencias.palabras_clave(palabra) values(?)',
              { replacements: [palabra] }
            )).insertId;
        // asociar
        await tx.query(
          'insert into residencias.reporte_palabras(proyecto_id, palabra_id)
values(?,?)',
          { replacements: [report_id, palabra_id] }
        );
```

```
      }
      return report_id;
    });
  }

  async getReports() {
    const [rows, error] = await catchError(ReportsRepository.getReports());
    if (error) throw error;
    return rows;
  }

  async updateReport(data, file) {
    return await sequelize.transaction(async (tx) => {
      // decidir ruta PDF
      const pdf_route = file ? file.path.replace(/^.*reportes/, 'reportes')
: data.pdf_route;
      // actualizar reporte
      const [, err2] = await catchError(
        ReportsRepository.updateReport({ ...data, pdf_route }), tx
      );
      if (err2) throw err2;
      // limpiar y volver a insertar palabras
      await tx.query(
        'delete from residencias.reporte_palabras where proyecto_id = ?',
        { replacements: [data.report_id] }
      );
      for (const palabra of data.palabras_clave || []) {
        // mismo flujo de crear/obtener palabra y asociar...
      }
    });
  }

  async deleteReport({ report_id }) {
    const [, error] = await catchError(ReportsRepository.deleteReport({
report_id }));
    if (error) throw error;
  }
}

export default new ReportsService();
```

Para explicarlo más a detalle, al momento que mandamos llamar al service, el controlador ya guardó el archivo pdf que vamos a subir y nos pasa su path y lo que hacemos es normalizar la ruta para almacenarlo en la

carpeta reportes, después de eso llamamos a reportsRepository que invocará la procedure de créate_report , una vez hecho todo asociamos las palabras clave que se adjuntaron en la misma transacción (de sequelize no de mariadb), si no existe mete la nueva palabra (aquí no pude re usar la de créate keyword pero igual funciona con ese método local), y finalmente relaciona la palabra con el reporte, el **tx** es sequelize transaction y lo usamos para hacer pasos extra que serán necesarios como asociar la palabra clave al reporte (lo manejamos por separado pero se juntan en reports_keywords, lo mismo pasa en update, empieza una transaction, decide si usar el pdf actual o uno nuevo, llama al repository, borra todas las filas de report-keywords y las vuelve a insertar al final de la transaction (esto a pesar de que puede estar generando muchos ids y demás, nunca afectará ya que la palbra siempre estará asociada a su reporte)

```
// controllers/reports.controller.js
import fs from 'fs';
import path from 'path';
import ReportsService from '../services/reports.service.js';

class ReportsController {
  createReport = async (req, reply) => {
    const data = await req.file();// { filename, file, ... }
    const body = req.body; // campos normales
    // mueve el stream a disco
    const target = path.join(__dirname, '..', 'reportes', data.filename);
    await pump(data.file, fs.createWriteStream(target));
    await ReportsService.createReport(body, { path: target });
    reply.code(201).sendSuccess({ message: 'Reporte creado correctamente'
});
  }

  getReports = async (req, reply) => {
    const rows = await ReportsService.getReports();
    reply.sendSuccess({ message: 'Reportes obtenidos', data: rows });
  }

  updateReport = async (req, reply) => {
    const { report_id } = req.params;
    const file = req.isMultipart ? await req.file() : null;
    if (file) {
      const target = path.join(__dirname, '..', 'reportes', file.filename);
      await pump(file.file, fs.createWriteStream(target));
      file.path = target;
```

```
    }
    await ReportsService.updateReport({ ...req.body, report_id }, file);
    reply.sendSuccess({ message: 'Reporte actualizado correctamente' });
  }

  deleteReport = async (req, reply) => {
    const { report_id } = req.params;
    await ReportsService.deleteReport({ report_id });
    reply.sendSuccess({ message: 'Reporte eliminado correctamente' });
  }
}

export default new ReportsController();
```

En reports controller el req.file proviene del mismo fastify nos permite leer un archivo del cuerpo de la petición y exponer su stream(

Pump mueve el fluj ode datos por el cuerpo HTTP a la carpeta que designamos para guardar todo, este controller no hace nada completo solamente.
Extra los input, guarda el pdf, llama al service y responde si todo bien o no.

```
import Joi from 'joi';

class ReportsValidator {
  createReport() {
    return Joi.object({
      student_name:    Joi.string().trim().required(),
      ctrlnumber:      Joi.string().trim().required(),
      major:           Joi.string().allow(null, '').optional(),
      report_title:    Joi.string().allow(null, '').optional(),
      company_id:      Joi.number().integer().required(),
      semestre:        Joi.number().integer().required(),
      palabras_clave:  Joi.array().items(Joi.string()).optional()
    });
  }

  getReports() {
    return Joi.object({});
  }
```

```
  updateReport() {
    return Joi.object({
      report_id:       Joi.number().integer().required(),
      student_name:    Joi.string().trim().required(),
      ctrlnumber:      Joi.string().trim().required(),
      major:           Joi.string().allow(null, '').optional(),
      report_title:    Joi.string().allow(null, '').optional(),
      company_id:      Joi.number().integer().required(),
      pdf_route:       Joi.string().optional(),
      semestre:        Joi.number().integer().required(),
      palabras_clave:  Joi.array().items(Joi.string()).optional()
    });
  }

  deleteReport() {
    return Joi.object({
      report_id: Joi.number().integer().required(),
    });
  }
}

export default new ReportsValidator();
```

```
import ReportsValidator from '../validators/reports.validator.js';
import { handleError } from './error.middleware.js';

class ReportsMiddleware {
  createReport = async (req, reply) => {
    try {
      req.body = await
ReportsValidator.createReport().validateAsync(req.body);
    } catch (err) {
      return handleError(err, req, reply);
    }
  };

  getReports = async (req, reply) => {
    try {
      req.query = await
ReportsValidator.getReports().validateAsync(req.query);
    } catch (err) {
      return handleError(err, req, reply);
    }
  };
```

```
  updateReport = async (req, reply) => {
    try {
      req.params = await
ReportsValidator.deleteReport().validateAsync(req.params);
      req.body   = await
ReportsValidator.updateReport().validateAsync(req.body);
    } catch (err) {
      return handleError(err, req, reply);
    }
  };

  deleteReport = async (req, reply) => {
    try {
      req.params = await
ReportsValidator.deleteReport().validateAsync(req.params);
    } catch (err) {
      return handleError(err, req, reply);
    }
  };
}

export default new ReportsMiddleware();
```

---

Users

```
import { QueryTypes } from 'sequelize';
import { sequelize } from '../config/database.js';

class UsersRepository {
  async createUser({ user_name, password, email, role_id }) {
    await sequelize.query(
      'call residencias.create_user(?,?,?,?);',
      { replacements: [user_name, password, email, role_id] }
    );
    return { user_name, password, email, role_id };
  }

  async getUsers() {
    const rows = await sequelize.query(
      'call residencias.get_users();',
      { type: QueryTypes.SELECT }
    );
    return rows;
```

```
  }

  async updateUser({ user_id, user_name, password, email, role_id, is_active
}) {
    await sequelize.query(
      'call residencias.update_user(?,?,?,?,?,?);',
      { replacements: [user_id, user_name, password, email, role_id,
is_active] }
    );
    return { user_id, user_name, password, email, role_id, is_active };
  }

  async deleteUser({ user_id }) {
    await sequelize.query(
      'call residencias.delete_user(?);',
      { replacements: [user_id] }
    );
    return { user_id };
  }

  async updateStatus({user_id, is_active}){
    await sequelize.query(
        'call residencias.update_user_status(?,?);',
        { replacements: [user_id, is_active]}
    );
    return {user_id, is_active};
  }

  async updatePassword({user_id, password}){
    await sequelize.query(
        'call residencias.update_user_password(?,?);',
        { replacements: [user_id, password]}
    );
    return {user_id, password};
  }
}

export default new UsersRepository();
```

```
import UsersRepository from '../repositories/users.repository.js';
import { catchError } from '../helpers/catch.error.js';

class UsersService {
  async createUser(data) {
```

```
    const [result, error] = await catchError(
      UsersRepository.createUser(data)
    );
    if (error) throw error;
    return result;
  }

  async getUsers() {
    const [result, error] = await catchError(
      UsersRepository.getUsers()
    );
    if (error) throw error;
    return result;
  }

  async updateUser(data) {
    const [result, error] = await catchError(
      UsersRepository.updateUser(data)
    );
    if (error) throw error;
    return result;
  }

  async deleteUser(data) {
    const [result, error] = await catchError(
      UsersRepository.deleteUser(data)
    );
    if (error) throw error;
    return result;
  }

  async updateStatus(data){
    const [result, error] = await catchError(
        UsersRepository.updateStatus(data)
    );
    if (error) throw error;
    return result;
  }

  async updatePassword(data){
    const [result, error] = await catchError(
        UsersRepository.updatePassword(data)
    );
    if (error) throw error;
    return result;
```

```
  }
}

export default new UsersService();
```

```
import UsersService from '../services/users.service.js';

class UsersController {
  createUser = async (req, reply) => {
    const { user_name, password, email, role_id } = req.body;
    const result = await UsersService.createUser({ user_name, password,
email, role_id });
    reply.code(201).sendSuccess({ message: 'User created', data: result });
  };

  getUsers = async (req, reply) => {
    const result = await UsersService.getUsers();
    reply.sendSuccess({ message: 'Users fetched', data: result });
  };

  updateUser = async (req, reply) => {
    const { user_id, user_name, password, email, role_id, is_active } =
req.body;
    const result = await UsersService.updateUser({ user_id, user_name,
password, email, role_id, is_active });
    reply.sendSuccess({ message: 'User updated', data: result });
  };

  deleteUser = async (req, reply) => {
    const { user_id } = req.params;
    const result = await UsersService.deleteUser({ user_id });
    reply.code(204).sendSuccess({ message: 'User deleted', data: result });
  };

  updateStatus = async (req, reply) => {
    const {user_id, is_active} = req.body;
    const result = await UsersService.updateStatus({user_id, is_active});
    reply.sendSuccess({message: 'User status updated', data: result});
  }

  updatePassword = async (req, reply) => {
    const {user_id, password} = req.body;
    const result = await UsersService.updatePassword({user_id, password});
```

```
    reply.sendSuccess({message: 'Password updated successfuly', data:
result});
  }
}

export default new UsersController();
```

```
import Joi from 'joi';

class UsersValidator {
  createUser() {
    return Joi.object({
      user_name: Joi.string().trim().required(),
      password:  Joi.string().trim().required(),
      email:     Joi.string().email().required(),
      role_id:   Joi.number().integer().required(),
    });
  }

  getUsers() {
    return Joi.object({});
  }

  updateUser() {
    return Joi.object({
      user_id:   Joi.number().integer().required(),
      user_name: Joi.string().trim().required(),
      password:  Joi.string().trim().required(),
      email:     Joi.string().email().required(),
      role_id:   Joi.number().integer().required(),
      is_active: Joi.boolean().required(),
    });
  }

  deleteUser() {
    return Joi.object({
      user_id: Joi.number().integer().required(),
    });
  }

  updateStatus(){
    return Joi.object({
        user_id: Joi.number().integer().required(),
        is_active: Joi.boolean().required(),
```

```
    });
  }

  updatePassword(){
    return Joi.object({
        user_id: Joi.number().integer().required(),
        password: Joi.string().required(),
    })
  }
}

export default new UsersValidator();
```

```
import UsersValidator from '../validators/users.validator.js';
import { handleError } from './error.middleware.js';

class UsersMiddleware {
  createUser = async (req, reply) => {
    try {
      req.body = await UsersValidator.createUser().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  getUsers = async (req, reply) => {
    try {
      req.query = await UsersValidator.getUsers().validateAsync(req.query);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  updateUser = async (req, reply) => {
    try {
      req.body = await UsersValidator.updateUser().validateAsync(req.body);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  deleteUser = async (req, reply) => {
    try {
```

```
      req.params = await
UsersValidator.deleteUser().validateAsync(req.params);
    } catch (err) {
      handleError(err, req, reply);
    }
  };

  updateStatus = async (req, reply) => {
    try {
        req.body = await
UsersValidator.updateStatus().validateAsync(req.body);
    } catch (err) {
        handleError(err, req, reply);
    }
  }

  updatePassword = async (req, reply) => {
    try{
        req.body = await
UsersValidator.updatePassword().validateAsync(req.body);
    } catch (err) {
        handleError(err, req, reply);
    }
  }
}

export default new UsersMiddleware();
```

---

Hasta el momento ya terminamos con el BE puro y ya probamos los endpoints en Postman ahora pasaremos a hacer una interfaz (frontend) manejando React.

Lo que sigue es hacer la lógica del login, authenticadores, javawebtoken, entre otros y ya con eso podríamos pasarnos al frontend.

---

## Frontend

---