Registro y login de usuarios

Antes de continuar con el proyecto, vamos a aplicar una parte del patrón de diseño MVC, en el cual separaremos nuestras rutas y nuestros controladores en dos módulos distintos.

Por lo cual, la estructura de nuestro proyecto quedará de la siguiente manera:

```
├── package.json
├── package-lock.json
├── .gitignore
├── src
│   ├── index.js
│   ├── services
│   │   └── container.js
│   ├── controllers
│   │   ├── auth.js
│   │   └── users.js
│   ├── routes
│   │   ├── auth.js
│   │   └── users.js
│   ├── models
│   │   └── user.js
│   ├── config
│   │   └── sequelize.js
│   └── utils
│       └── database.js
└── README.md
```

- Se puede observar que añadiremos dos nuevos directorios llamados `controllers` y `routes`, en los cuales se almacenarán los controladores y las rutas respectivamente.
- `controllers` contendrá los controladores de cada entidad, en este caso, `auth`, que tendrá las funciones necesarias para el login y el registro de usuarios.
- `routes` contendrá las rutas de cada entidad, en este caso, `auth`, que tendrá las rutas necesarias para el login y el registro de usuarios.
- Recordar exportar e importar los archivos necesarios para que el proyecto funcione correctamente.

Registro de usuarios

Para el registro de usuarios, vamos a crear una ruta en `routes/auth.routes.js` que se encargue de recibir los datos del usuario y enviarlos al controlador correspondiente.

```
const router = require("express").Router(); // Importamos el router de
express
const { createUser, loginUser } =
require("../controllers/auth.controllers"); // Importamos el controlador
de auth

// Crear una ruta de tipo POST para registrar un usuario en la base de
datos
router.post("/register", createUser); //
http://localhost:8080/auth/register

// Crear una ruta de tipo POST para iniciar sesión
router.post("/login", loginUser); // http://localhost:8080/auth/login

module.exports = router; // Exportamos el router para poder usarlo en
otros archivos
```

Además, será necesario configurar el grupo de rutas en `index.js`:

```
const express = require("express");
const Container = require("./services/container");
const { connectToDB } = require("./config/sequelize");

const authRoutes = require("./routes/auth.routes");

connectToDB(); // Conectamos a la base de datos
const container = new Container();

const app = express();

// Para leer los datos del body de la petición haremos la siguiente
configuración
app.use(express.json());

// configuramos el grupo de rutas /auth
app.use("/auth", authRoutes);

app.listen(8080, () => {
  console.log("Servidor escuchando en el puerto 8080");
});
```

Ahora, en el controlador `controllers/auth.controllers.js` vamos a crear las funciones necesarias para registrar un usuario en la base de datos:

```
const { response } = require("express");
const { User } = require("../models/db"); // Importamos el modelo User

const createUser = async (req, res) => {
  try {
    const newUser = await User.create(req.body); // Creamos el usuario en
la base de datos
    console.log(newUser);

    return res.status(201).json({
      // 201 para indicar que se ha creado correctamente
      ok: true,
      msg: "Usuario creado correctamente",
      data: newUser,
    });
  } catch (error) {
    console.log(error);
    return res.status(500).json({
      ok: false,
      msg: "Error al crear el usuario",
      data: null,
    });
  }
};


module.exports = {
  createUser,
};
```

Encriptando la contraseña

Como podemos apreciar en el código anterior, estamos almacenando la contraseña del usuario en la base de datos, lo cual no es una buena práctica de seguridad, por lo cual, vamos a encriptar la contraseña antes de almacenarla en la base de datos.

Para ello, vamos a instalar la librería `bcryptjs`, la cual es una librería que se encarga de encriptar y desencriptar cadenas de texto.

```
npm install bcryptjs
```

Ahora, en el controlador `controllers/auth.controllers.js` vamos a importar la librería y a modificar la función `createUser`:

```
const { response } = require("express");
const bcrypt = require("bcryptjs");
const { User } = require("../models/db"); // Importamos el modelo User

const createUser = async (req, res) => {
  try {

    // Encriptamos la contraseña
    const salt = bcrypt.genSaltSync(10); // Generamos el salt
    // Nota: salt es un valor aleatorio que se agrega a la contraseña
para que el hash sea más seguro
    req.body.password = bcrypt.hashSync(req.body.password, salt); //
Encriptamos la contraseña

    const newUser = await User.create(req.body); // Creamos el usuario en
la base de datos
    console.log(newUser);

    // Antes de devolver los datos del usuario, eliminamos el campo
password
    newUser.password = undefined;

    return res.status(201).json({
      // 201 para indicar que se ha creado correctamente
      ok: true,
      msg: "Usuario creado correctamente",
      data: newUser,
    });
  } catch (error) {
    console.log(error);
    return res.status(500).json({
      ok: false,
      msg: "Error al crear el usuario",
      data: null,
    });
  }
};


module.exports = {
  createUser,
};
```

- Nota: en este caso, estamos encriptando la contraseña de forma síncrona, pero lo recomendable es hacerlo de forma asíncrona, ya que si la contraseña es muy larga, puede tardar mucho tiempo en encriptarse. Sin embargo, para este ejemplo, lo haremos de forma síncrona.

Login de usuarios

Para el login de usuarios, vamos a crear una ruta en `routes/auth.routes.js` que se encargue de recibir los datos del usuario y enviarlos al controlador correspondiente.

```
const router = require("express").Router(); // Importamos el router de
express
const { createUser, loginUser } =
require("../controllers/auth.controllers"); // Importamos el controlador
de auth

// Crear una ruta de tipo POST para registrar un usuario en la base de
datos
router.post("/register", createUser); //
http://localhost:8080/auth/register

// Crear una ruta de tipo POST para iniciar sesión
router.post("/login", loginUser); // http://localhost:8080/auth/login

module.exports = router; // Exportamos el router para poder usarlo en
otros archivos
```

Además, será necesario crear la función `loginUser` en el controlador `controllers/auth.controllers.js`:

Para poder efectuar la autenticación, vamos a seguir los siguientes pasos:

1. Buscar el usuario en la base de datos

Vamos a usar un método de Sequelize llamado `findOne`, el cual nos permite buscar un registro en la base de datos que cumpla con ciertas condiciones.

```
  User.findOne({
    where: {
      email: req.body.email,
    },
  });
```

findOne recibe un objeto con las condiciones de búsqueda, en este caso, buscamos un usuario cuyo email sea igual al email que recibimos en la petición.
Si no se encuentra ningun usuario con ese email, el método devolverá `null`, que como sabemos en javascript un `null` se puede interpretar como un `false`.

2. Verificar la contraseña

Si el usuario existe, vamos a verificar que la contraseña sea correcta, para ello, vamos a usar el método `compareSync` de la librería `bcryptjs`, el cual recibe dos parámetros, el primero es la contraseña que queremos verificar y el segundo es la contraseña encriptada que tenemos almacenada en la base de datos.

```
  const isPasswordValid = bcrypt.compareSync(
    req.body.password,
    user.password
  );
```

compareSync devuelve un booleano, el cual será `true` si la contraseña es correcta y `false` si no lo es.

```
const { response } = require("express");
const bcrypt = require("bcryptjs");
const { User } = require("../models/db"); // Importamos el modelo User

const createUser = async (req, res) => {
  try {
    // Encriptamos la contraseña
    const salt = bcrypt.genSaltSync(10); // Generamos el salt
    // Nota: salt es un valor aleatorio que se agrega a la contraseña
para que el hash sea más seguro
    req.body.password = bcrypt.hashSync(req.body.password, salt); //
Encriptamos la contraseña

    const newUser = await User.create(req.body); // Creamos el usuario en
la base de datos
    console.log(newUser);

    // Antes de devolver los datos del usuario, eliminamos el campo
password
    newUser.password = undefined;

    return res.status(201).json({
      // 201 para indicar que se ha creado correctamente
      ok: true,
      msg: "Usuario creado correctamente",
      data: newUser,
    });
  } catch (error) {
    console.log(error);
    return res.status(500).json({
      ok: false,
      msg: "Error al crear el usuario",
      data: null,
```

```
    });
  }
};

const loginUser = async (req, res) => {
  try {
    // 1. Buscamos el usuario en la base de datos
    const user = await User.findOne({
      where: {
        email: req.body.email,
      },
    });

    // Si no existe el usuario
    if(!user){
        return res.status(400).json({
            ok: false,
            msg: 'El usuario no existe',
            data: null
        });
    }

    // 2. Comparamos la contraseña
    const validPassword = bcrypt.compareSync(req.body.password,
user.password); // Comparamos la contraseña

    // Si la contraseña no es válida
    if(!validPassword){
        return res.status(400).json({
            ok: false,
            msg: 'Contraseña incorrecta',
            data: null
        });
    }

    // 3. Generamos el token  TODO: Generar el token

    // 4. Devolvemos los datos del usuario
    user.password = undefined; // Eliminamos el campo password

    return res.status(200).json({
        ok: true,
        msg: 'Inicio de sesión correcto',
        data: user
    });


  } catch (error) {
    console.log(error);
```

```
    return res.status(500).json({
      ok: false,
      msg: "Error al iniciar sesión",
      data: null,
    });
  }
};

module.exports = {
  createUser,
  loginUser,
};
```

- Nota: en este caso, estamos comparando la contraseña de forma síncrona, pero lo recomendable es hacerlo de forma asíncrona, ya que si la contraseña es muy larga, puede tardar mucho tiempo en compararse. Sin embargo, para este ejemplo, lo haremos de forma síncrona.