Contenido

[1. Configurar el backend con Node.js y Express.js 1](#_Toc140855923)

[a. Crea una nueva carpeta para tu proyecto y ábrela en tu editor de código. 1](#_Toc140855924)

[b. Instala los paquetes necesarios 1](#_Toc140855925)

[c. Configura Express.js en server.js e importa las dependencias necesarias: 2](#_Toc140855926)

[2. Configurar la base de datos MongoDB 2](#_Toc140855927)

[a. configura la conexión con MongoDB 2](#_Toc140855928)

[3. Crear los modelos de datos 3](#_Toc140855929)

[4. Implementar las rutas y controladores 4](#_Toc140855930)

[a. Controladores 4](#_Toc140855931)

[b. Routes 8](#_Toc140855932)

Crear un modelo de base de datos, relacional en mi caso

# Configurar el backend con Node.js y Express.js

## Crea una nueva carpeta para tu proyecto y ábrela en tu editor de código.

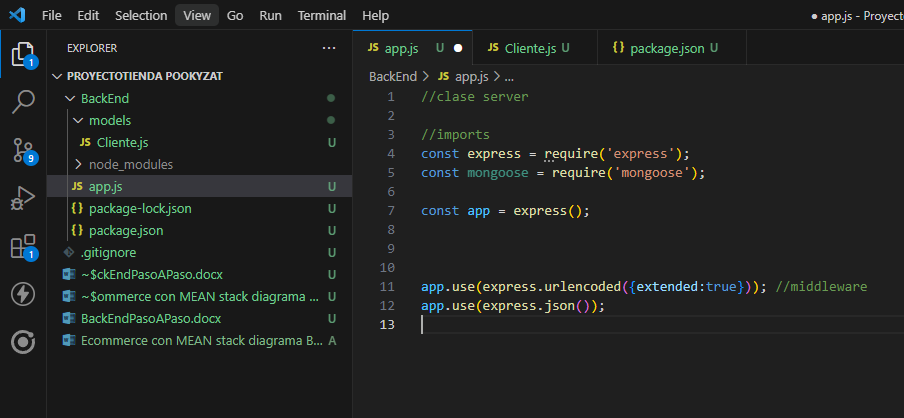
Inicializa un nuevo proyecto de Node.js ejecutando npm init en la terminal y siguiendo las instrucciones.

## **Instala los paquetes necesarios**

ejecutando npm install express mongoose en la terminal.

## Configura Express.js en server.js e importa las dependencias necesarias:

1. //clase server
2. //imports
3. const express = require('express');
4. const mongoose = require('mongoose');
5. const app = express();
7. app.use(express.urlencoded({extended:true})); //middleware
8. app.use(express.json());



# Configurar la base de datos MongoDB

Asegúrate de tener MongoDB instalado y ejecutándose.

## configura la conexión con MongoDB

En el archivo app.js del backend, configura la conexión con MongoDB utilizando Mongoose.

//coneccion a la base de datos

const uri='mongodb://127.0.0.1:27017/Pookyzat'

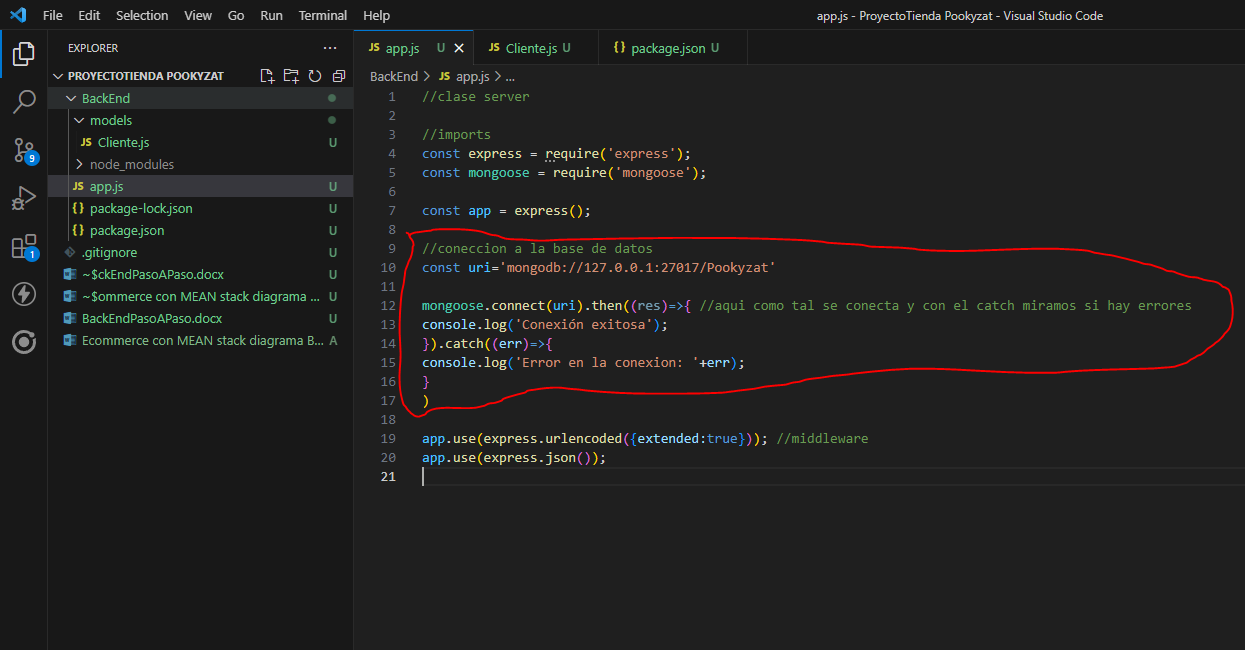
mongoose.connect(uri).then((res)=>{ //aqui como tal se conecta y con el catch miramos si hay errores

console.log('Conexión exitosa');

}).catch((err)=>{

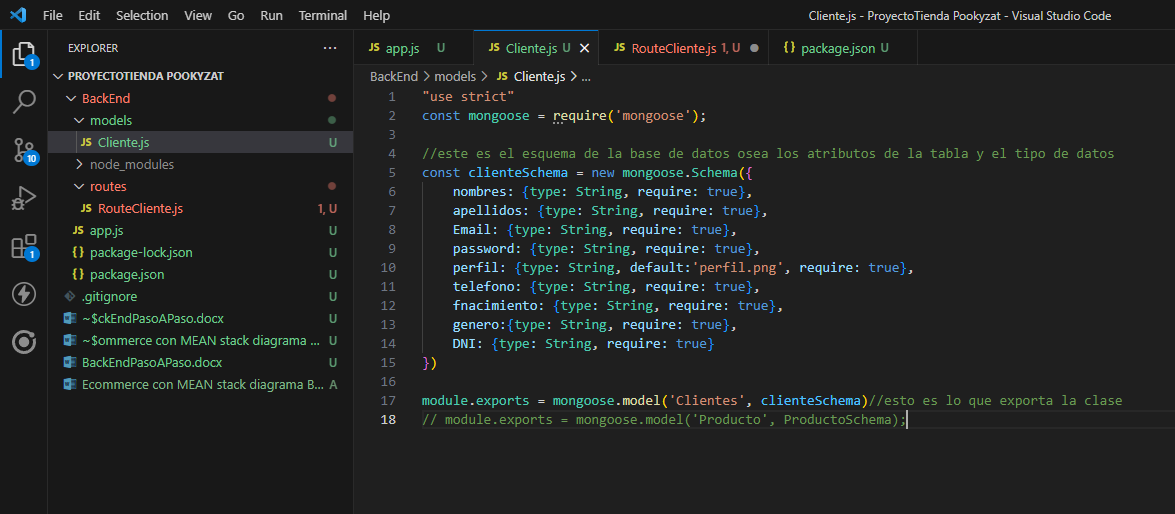
console.log('Error en la conexion: '+err);

}



# Crear los modelos de datos

Crea los schemas con mongoose en una carpeta llamada models



"use strict"

const mongoose = require('mongoose');

//este es el esquema de la base de datos osea los atributos de la tabla y el tipo de datos

const clienteSchema = new mongoose.Schema({

    nombres: {type: String, require: true},

    apellidos: {type: String, require: true},

    Email: {type: String, require: true},

    password: {type: String, require: true},

    perfil: {type: String, default:'perfil.png', require: true},

    telefono: {type: String, require: true},

    fnacimiento: {type: String, require: true},

    genero:{type: String, require: true},

    DNI: {type: String, require: true}

})

module.exports = mongoose.model('Clientes', clienteSchema)//esto es lo que exporta la clase

# Implementar las rutas y controladores

## Controladores

En la carpeta controller creamos ClienteController.js y creamos las funciones de login y registrarCliente.

'use strict'

const Cliente = require('../models/Cliente');

const bcrypt = require('bcrypt');//lo recomendable es encriptar strings menores a 72 bytes

const jwt= require('jsonwebtoken');

var moment = require('moment');

const secret = 'pookymesa' //no se si esto va en otro lado mas seguro

async function registrarCliente(req, res){//creamos una funcion async(porque tenemos que esperar respuesta) la cual va a recibir req:que es donde viene el body en req.body y una res, que es la respuesta http de la peticion

    try {

        const dataBody=req.body; //guardamos body en una constante

        //verificamos si existe el cliente

        const existeCliente = await Cliente.findOne({email:dataBody.email})//utilizamos la funcion findOne porque si no encuentra al cliente da null si usaramos find() nunca daria null esta funcion recibe un objeto poner {email: }

        if(existeCliente){

            return res.status(400).json({ message: 'El cliente ya existe' });

        }else{

            if(dataBody.password){

                // Hashear la contraseña antes de guardarla

                bcrypt.hash(dataBody.password, 10, async function(err, hash) {

                    if(hash){

                        dataBody.password=hash;

                        const clienteNuevo= await Cliente.create(dataBody);//se crea el cliente y guardamos este dato en una constante para usarlo mas adelante

                    res.status(200).json({data:clienteNuevo})

                    }else{

                        console.log(err);

                        res.status(200).send({message:'ErrorServer', error:err})

                    }

                });

            }else{

                return res.status(400).send({ message: 'Ingrese una contraseña porfavor' });

            }

        }

    } catch (error) {

        console.log(error);

        res.status(500).send({ message: 'Error al hacer login', error: error });

    }

}

async function loginClientes(req,res){

    try {

        const body = req.body;

        if(body.email && body.password){

            //ecnontramos que el email exista

            const user = await Cliente.findOne({email:body.email});//aqui se encuentra el email dentro de la coleccion y trae todo el objeto

            if(!user){

                return res.status(401).json({ message: 'Email incorrecto' });

            }else{

                //verificamos contraseña

                // console.log(user);

                bcrypt.compare(body.password, user.password, function(err,result){

                    if(result){

                        //creamos el token

                        const payload = {

                            sub: user.\_id,

                            nombre: user.nombre,

                            apellido: user.apellido,

                            email: user.email,

                            rol:user.rol,

                            iat: moment().unix(),//información de fecha y hora cuando el JWT fue usado;

                            exp: moment().add(7,'days').unix()//se usa unix porque se maneja en segundos

                        }

                        const token = jwt.sign(payload, secret);

                        //devolvemos el token

                        res.status(200).json({data: token})

                    }else{

                        console.log(err)

                        return res.status(401).send({ message: 'Contraseña incorrecta'});

                    }

                })

            }

        }else{

            return res.status(400).json({ message: 'Ingrese email y contraseña' });

        }

    } catch (error) {

        console.log(error)

        res.status(500).json({ message: 'Error al hacer login', error });

    }

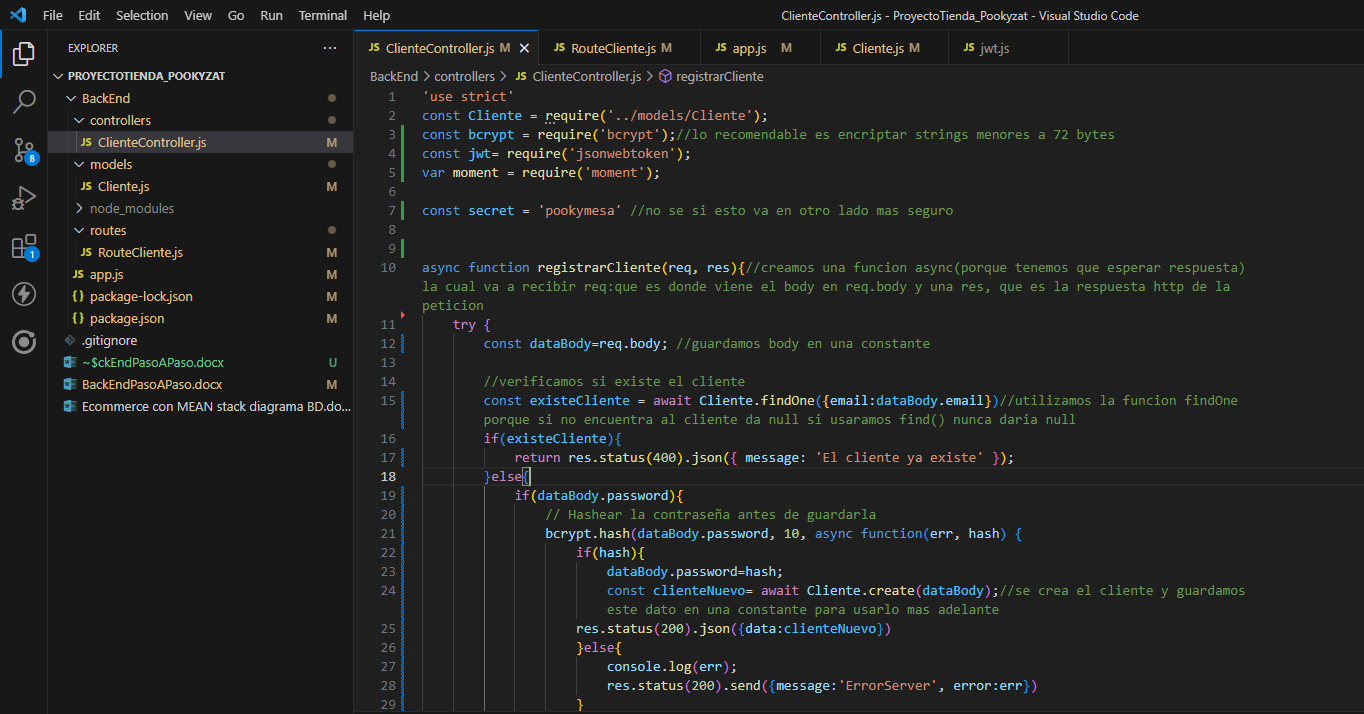
}

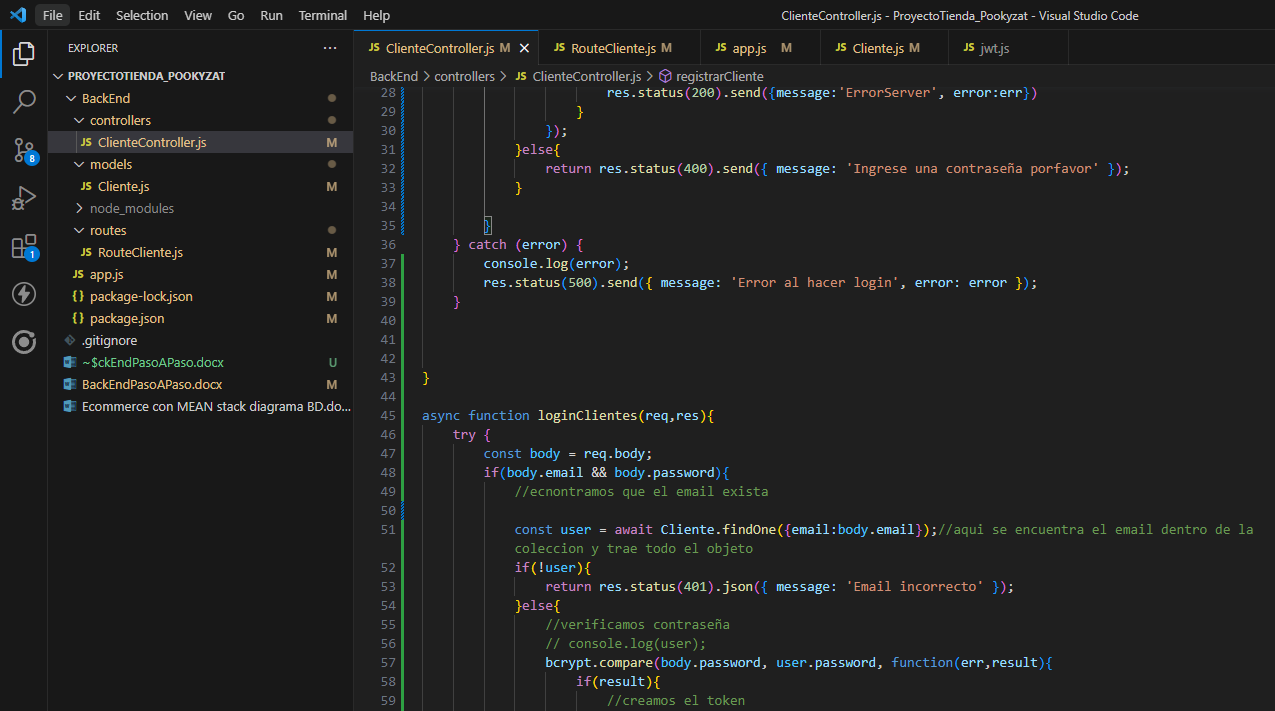
module.exports={

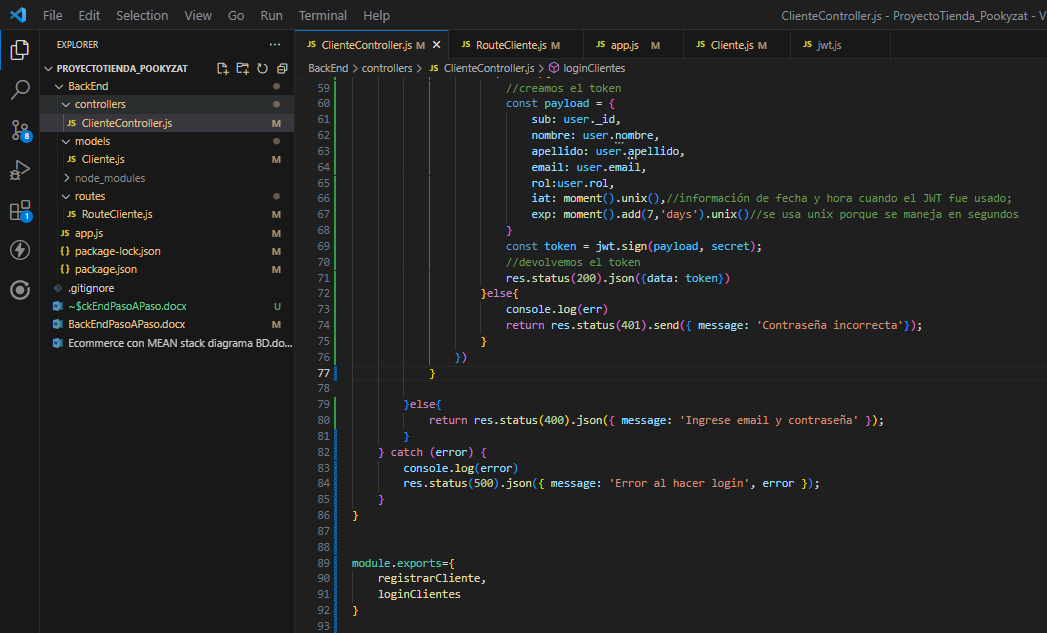
    registrarCliente,

    loginClientes

}







## Routes

Creamos la ruta para cada función de nuestro controlador, creamos la carpeta routes y el archivo RouteCliente.js

"use strict"

const ClienteController = require('../controllers/ClienteController');

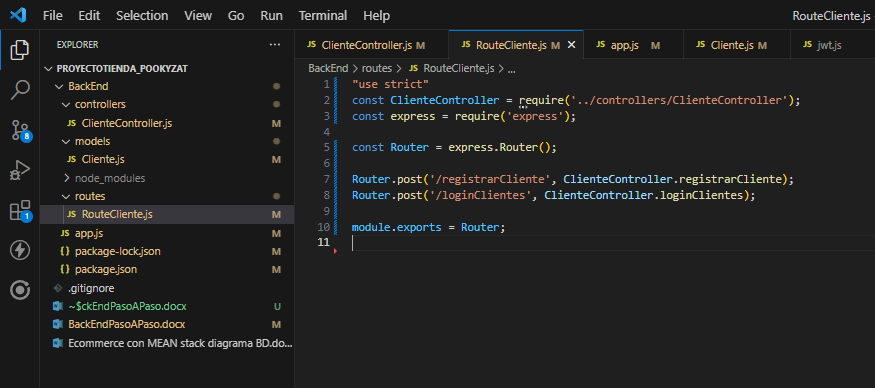
const express = require('express');

const Router = express.Router();

Router.post('/registrarCliente', ClienteController.registrarCliente);

Router.post('/loginClientes', ClienteController.loginClientes);

module.exports = Router;



//Primer paso a paso

Aquí tienes un paso a paso para crear un CRUD utilizando el stack MEAN (MongoDB, Express.js, Angular, Node.js):

Paso 1: Configurar el backend con Node.js y Express.js

* Crea una nueva carpeta para tu proyecto y ábrela en tu editor de código.
* Inicializa un nuevo proyecto de Node.js ejecutando **npm init** en la terminal y siguiendo las instrucciones.
* Instala los paquetes necesarios ejecutando **npm install express mongoose** en la terminal.
* Crea un archivo **index.js** (o cualquier otro nombre) para tu backend y configura un servidor Express.js básico con las rutas y controladores para el CRUD.

Paso 2: Configurar la base de datos MongoDB

* Asegúrate de tener MongoDB instalado y ejecutándose.
* En el archivo **index.js** del backend, configura la conexión con MongoDB utilizando Mongoose.

Paso 3: Crear los modelos de datos

* Define los esquemas de tus datos en Mongoose, que representarán los documentos de MongoDB.
* Crea los modelos utilizando los esquemas definidos, los cuales te permitirán interactuar con la base de datos.

Paso 4: Implementar las rutas y controladores

* Crea las rutas para las operaciones CRUD (crear, leer, actualizar, eliminar) en el archivo **index.js**.
* Define los controladores correspondientes para cada ruta, donde se ejecutarán las acciones en la base de datos.

Paso 5: Configurar el frontend con Angular

* Crea una nueva carpeta para el frontend en tu proyecto y ábrela en tu editor de código.
* Instala Angular CLI globalmente ejecutando **npm install -g @angular/cli** en la terminal.
* Crea un nuevo proyecto de Angular ejecutando **ng new nombre-proyecto** en la terminal y sigue las instrucciones.
* Navega al directorio del proyecto Angular con **cd nombre-proyecto**.

Paso 6: Crear componentes y servicios

* Crea los componentes necesarios para las vistas de tu CRUD, como lista de elementos, formulario de creación/edición, etc.
* Crea servicios Angular para interactuar con el backend, utilizando HTTP para realizar las solicitudes al servidor.

Paso 7: Configurar las rutas y enlaces

* Configura las rutas en Angular para mapear las URL a los componentes correspondientes.
* Crea enlaces en tus vistas para navegar entre las diferentes páginas y acciones del CRUD.

Paso 8: Implementar las funcionalidades del CRUD en el frontend

* Utiliza los servicios Angular para realizar las solicitudes al backend y realizar las operaciones CRUD (crear, leer, actualizar, eliminar) en los componentes correspondientes.

Paso 9: Ejecutar la aplicación

* Inicia el servidor backend ejecutando **node index.js** en la terminal en la carpeta del backend.
* Inicia la aplicación Angular ejecutando **ng serve** en la terminal en la carpeta del frontend.
* Abre tu navegador y accede a **http://localhost:4200** para ver y probar tu aplicación CRUD.

¡Felicidades! Ahora tienes un CRUD básico utilizando el stack MEAN. Puedes continuar añadiendo más funcionalidades, validaciones y mejoras según tus necesidades.