# Introduccion

Aplicación de notas con autenticación de usuarios y rutas privadas

# Librerias / modulos

**Express** para el servidor

**Connect-flash** mensajes entre vistas, guarda la info en cookie o en sesión

**Bcryptjs** módulo de cifrado (como contraseñas)

**Express-handlebars** ViewEngine

**Express-session** guardar información en la memoria del server

**Method-override** enviar peticiones put o delete desde la vista

**Mongoose** manejar la bd mongo

**Passport** autenticación del usuario

**Passport-local** para usar bd local de autenticación (se puede usar Passport para autenticar con twitter, fb, google…)

## Modulos dev

**Dotenv** usar variables de entorno

**Nodemon** reinicia el server automáticamente

**Npm-check-updates** valida si hay actualizaciones

# Estructura

**Database.js** configuración de la base de datos

**Server.js** donde estará express

**Controllers** controladores

**Public** archivos estáticos como css

**Views** vistas

**Helpers** para ayudar a handlebars con los bucles y condicionales

## Server.js

Se crea el servidor y se exporta para usarse en index

const express = require('express');

const app = express(); //se crea el servidor

module.exports = app;

## index.js

Se importa el server

const app = require('./server');

dotenv

require('dotenv').config();

Con esto, buscara un archivo llamado .env, lo leerá y lo asignará a la variable de entorno, para acceder a esta variable se ejecuta process.env.NOMBRE\_VARIABLE

require('./database'); //se ejecuta la BD

## Method override

Para poder usar el método delete, o put (router.delete), y poderlo enviar desde un formulario, se debe usar el middleware de method-override

//delete notes

router.delete("/notes/delete/:id", deleteNote);

Middleware

app.use(methodOverride('\_method'));

En el form que va a ir el método delete, debe ir de la siguiente manera:

 <form action="/notes/delete/{{\_id}}?\_method=DELETE" method="POST">

Y se debe agregar un input de tipo hidden en el formulario

          <form action="/notes/delete/{{\_id}}?\_method=DELETE" method="POST">

                <input type="hidden" name="\_method" value="DELETE">

## Connect-flash – Express session

Nos permite enviar mensaje entre vistas, se guardan en express sesión

Se deben importar en el server

const flash = require('connect-flash');

const session = require('express-session')

Y usarse como middleware

app.use(session({

    secret: 'Secretoseguro25448',

    resave: 'true',

    saveUninitialized: 'true'

}));

app.use(flash());

En el controlador, creamos el mensaje: req.flash

notesController.createNewNote= async (req,res)=>{

    //extraemos titulo y descripcion de la nota

    const {tittle, description} = req.body;

    //creamos una nota nueva, previamente importada, y le asignamos los valores del body, js asigna automaticamente los valores sin necesidad de escribir description: description

    const newNote = new Note({tittle, description});

    await newNote.save();//mongoose guarda la información en la bd

    req.flash('success\_mg', ' Note added suuccesfully'); //creamos el mensaje

    res.redirect('/notes')

};

En cada controlador está la ruta, pero para que se pueda acceder desde toda la aplicación a la variable success\_mg se define como variable global

app.use((req, res, next) =>{//el next se usa para que continue ejecutando lo que hay debajo

    res.locals.success\_msg = req.flash('success\_msg');

    next();

});

Cuando ejecuta el código, la variable **success\_mg** Obtiene el valor (note added… en este caso), y accedemos a ella

Se agrega un .hbs en partials, que valida si existe el mensaje, si si, lo muestra

{{#if success\_msg}}

<div class="alert alert-success alert-dismissible fade show">

    <strong> {{success\_msg}} </strong>

    <button type="button" class="close" data-bs-dismiss="alert">&times;</button>

</div>

{{!-- se crea el botón para cerrar el cuadro de dialogo, ell &times es para mostrar una x de cerrar --}}

{{/if}}

## Passport

Se encarga de las sesiones de los usuarios, permite logear con google, twitter… por ese motivo se instala también Passport-local

En este caso se usara como middleware

Se importa

const passport = require('passport');

const LocalStrategy = require('passport-local').Strategy;

y se definen los campos que voy a recibir

passport.use(new LocalStrategy({

    usernameField: 'email',

    passwordField: 'password'

}))

const passport = require('passport');

const LocalStrategy = require('passport-local').Strategy;

const User = require('../models/user'); //se importa porque model puede interactuar con la BD

passport.use(new LocalStrategy({

    // Se reciben los datos

    usernameField: 'email',

    passwordField: 'password'

}, async (email, password, done)=>{

    // Se validan los datos

    const user = await User.findOne({email});

    if(!user){

        // Se valida el correo del usuario

        // si no existe, acaba la funcion y con la funcion done le pasamos el error, se integra con connect flash

        return done(null, false, {message: 'Not User Found'});

    }else{

        // Se revisa la contraseña

        const match = await user.matchPassword(password); //esta es la clave recibida en el form, y la revisa con el de la bd, devuelve true/false

        if (match){

            return done(null, user); //no hay error y devuelve el usuario

        }else{

            return done(null, false, {message: 'incorret password'});

        }

    }

}));

passport.serializeUser((user, done) =>{

    //Recibe la funcion con el usuario(id)

    done(null, user.id)

});

passport.deserializeUser((id, done)=>{

    //se comprueba cada vez que el usuario esta navegando, a través del id, si tiene los permisos

    User.findById(id, (error, user =>{

        done(err, user);//si recibe error, termina con el error, sino con el usuario

    }))

})

Se importa en el server, y se ubica luego de sesión

const passport = require('passport')

require('./config/passport')

app.use(passport.initialize());

app.use(passport.session());

En el controlador lo importamos

const passport = require('passport');

usersController.signIn= passport.authenticate('local', {//se le pasa el nombre por defecto local, y el objeto

    //si hay error

    failureRedirect: '/users/signin',

    successRedirect: '/notes',

    failureFlash: true //si hay error, envia mensaje en flash

})

En connect flash, en las variables globales del servidor, se agrega el mensaje de error que eniva Passport

res.locals.error = req.flash('error');

y lo definimos en partials

{{#if error}}

    <div class="alert alert-danger alert-dismissible fade show">

    <strong> {{error}} </strong>

    <button type="button" class="close" data-bs-dismiss="alert">&times;</button>

</div>

{{/if}}

SI el usuario está autenticado, existe una variable global

res.locals.user = req.user || null;

### Mostrar notas si esta logeado

Si esta logeado muestra el menú de notas, si no, muestra el menú de login

        {{#if user}}

          <li class="nav-item dropdown">

            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

              Notes

            </a>

            <div class="dropdown-menu" aria-labelledby="navbarDropdown">

              <a class="dropdown-item" href="/notes">All Notes</a>

              <a class="dropdown-item" href="/notes/add">Add A Note</a>

              <div class="dropdown-divider"></div>

              <a class="dropdown-item" href="/users/logout">Logout</a>

            </div>

          </li>

        {{else}}

          <li class="nav-item">

            <a class="nav-link" href="/users/signin">Login</a>

          </li>

          <li class="nav-item">

            <a class="nav-link" href="/users/signup">Register</a>

          </li>

        {{/if}}

### Logout

Passport tiene un método

usersController.logout = (req, res)=>{

    req.logout();

    req.flash('success\_msg', 'Loged out');

    res.redirect('/users/signin');

}