Empezando

Creo la carpeta del proyecto y lo inicio:

$npm init –y

Inicio GIT:

$git init

Instalo los modulos:  
$npm install express express-handlebars express-session

Express-handlebars: ayudara a implementar un motor de plantillas(una manera de extender el html con bucles y condicionales). Al final termina siendo html

<https://www.npmjs.com/package/express-handlebars>

express-session: para sesión de usuario

<https://www.npmjs.com/package/express-session>

method-override: normalmente los formularios permiten enviar post y get, pero existen otros métodos como put o delete

mongoose: es un módulo que permite conectar el servidor con una base de datos mongo

<https://mongoosejs.com/>

Passport y Passport-local: para autenticar usuarios en DB

<https://www.npmjs.com/package/passport>

<https://www.npmjs.com/package/passport-local>

bcryptjs: para cifrar la constrase;a, aplica un algoritmo a los string y lo convierte en hash

<https://www.npmjs.com/package/bcryptjs>

connect-flash: para enviar mensajes de: usuario incorrecto, etc

<https://www.npmjs.com/package/connect-flash>

*$ npm install express express-handlebars express-session method-override mongoose passport passport-local bcryptjs connect-flash*

Luego hago commit a esto para iniciar. Ahora creo una estructura de carpetas:

Views: para poner html

Routes: url o rutas: login, register, about, etc

Public: imágenes, css, js, fuentes

Models: definir como lucirán los datos que iran a la bd, el eschema por asi decirlo

Helpers: para modulos utilitarios

Config: multiples archivos de configuración, bd, variables de entorno, autenticación

## Backend

Creando y configurando servidor en index.js, antes instalo nodemon para ayudarme al desarrollo:

Nota: para ejecutar nodemon debe ser por medio de un script o instalarlo -g(global).

Cree los archivos básicos de Routes, database

## Views (templates engine)

En este proyecto se esta usando handlebars por lo cual en vez de html tendremos .hbs para ello creo dentro de views/about.hbs e index.hbs y en routes le especifico que ya no muestre solo un texto sino que muestre la pagina creada.

Debo recordar que en la configuración de handlebars elegi main.hbs como template general que contendrá header, contenido, footer, etc. Este será mi container de las demás vistas.

Creo las vistas para users/signin y signup y cambio las rutas también.

Ahora voy a estilizar el main.hbs, voy al sitio de Bootstrap y copio link css y js. En la mayoría de etiquetas estoy usando clases Bootstrap.

También puedo crear mis propios CSS en public/css

## CRUD

Crear todas las vistas relacionadas con las notas, agregar, eliminar, actualizar y listar.

Voy a trabajar en el Routes/users.js y crear los Views necesarios en Views/notes/

En primer momento creo el archivo para nuevas notas add-note.hbs donde agregare el formulario estilizado desde donde enviare una nueva nota con POST, mi archivo routes/notes.js la recibe y la procesa. Para guardar en la bd necesito un Schema(esto es una especie de clase) q creare como Notes.js que usare en la ruta/notes.js Creado el modelo ahora tengo q importar el schema en routes/notes para usarlo y guardar el schema en la bd. usando POST(notes/new-note como función asíncrona y ciertas validaciones.

Luego de guardar redireccionare a donde se encuentran TODAS las notas, para ello también tengo q acceder a la bd por medio del schema Note.js. Necesitare una nueva view para ello (notes/all-notes).

Luego de maquetar all-notes también agregare el css de fontAwesome para usar iconos en el icono editar.

Para editar una nota lo que hago es al momento de renderizar la nota en all-notes agregar una nueva ruta con el ID de cada nota.

Creo la view edit-note con formulario y le paso la nota q quiero editar desde notes.js

Ahora cuando yo edite la nota y le de SAVE, tengo q enviar eso a otra ruta con los datos y mediante POST:

<form action="/notes/edit-note/{{note.\_id}}" method="POST" >

Podria desde notes.js usar el método POST para actualizar los datos pero usare PUT. Para ello pondré un input oculto:

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

Para usar este input tenemos un Middleware en index.js:

//en el form usare un input oculto(\_method)

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

con esto tengo que modificar mi URL para incluir una consulta (?\_method=PUT) aunque pasemos POST se reconocerá PUT :

<form action="/notes/edit-note/{{note.\_id}}?\_method=PUT" method="POST" >

Ahora uso put en notes.js y realizo la consulta para actualizar por id.

Ahora hare delete y de igual manera usare un input oculto:

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

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

<button type="submit" class="btn btn-danger btn-block btn-sm">Delete</button>

</form>

Y uso .delete(ruta,callback).

Para completar el CRUD usare ‘connect-flash’ que pasara mensajes entre vistas:

Pero necesito tener variables globales para hacerlo

app.use(flash())

//##########Gloval variables##########

app.use((req,res,next)=>{

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

res.locals.error\_msg=req.flash('error\_msg')

next()

})

Lo agrego en notes.js

//ahora ALMACENO ese nuevo dato en la BD pero de forma asincrona

await newNote.save()

//Para usar flash() y pasar mensajes

req.flash('success\_msg','Note added Successfully')

Luego creo un “partial” message para desplegarlo en un alert

## Register/SingUp

Ahorita se están creando notas para Todos, pero yo quiero que cada usuario cree notas y estas no sean vistas por otros.

Ahora trabajare en routes/users.js y las view de users. He trabajado primero en el formulario de SignUp ahora cuando se presione el botón enviara a otra ruta entonces voy a users.js

Trabajado en el método post que viene de signup y validado campos ahora debo GUARDAR, pero para ello necesito otro Schema/Users.js ahí creo el schema con los campos requeridos para un usuario además de dos métodos para encriptar el pass y para hacer match.

## Login/SingIn

Lo primero que hago es crear la interfaz de la view. Para la autenticación usare Passport y Passport-local ya que estos guardaran la sesión para no estar pidiendo a cada momento(localStorage). Para ello creo en config/Passport.js luego de esto debo hacer algunas configuraciones en index.js para loggerse:

const passport=require('passport')

//debe ir despues de session

app.use(passport.initialize())

//le digo a passport q use session de express

app.use(passport.session())

//flash va al final xq el login va primero

app.use(flash())

Ya está configurado, pero aún no estamos usando nuestra configuración siempre en index.js:

//importo el archivo config/passport.js

require('./config/passport.js')

Ahora voy a routes/users.js

//para AUTENTICAR usare passport, local se da por new LocalStrategy

router.post('/users/signin',passport.authenticate('local',{

successRedirect:'/notes',

failureRedirect:'/users/signin',

failureFlash:true//poder enviar mensajes

}))

No se están mostrando los errores, para ello voy a index.js y configuro variables globales

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

Ahora el partial

## Navigation

Creo un partial/navigation.hbs y reemplazo el texto que tenía. Ahora en el archivo usare código de Bootstrap y solo cambio dirección de rutas y nombres

Cree nueva ruta users/logout entonces voy a users.js y creo el método.

Pero aun loggeando y logout aun puedo acceder a todas las notas, debo asegurarlas, para ello creo un helper/auth.js y la utilizo, como lo que quiero asegurar es Notes, importo ese método

//para ASEGURAR esta ruta importo auth

const {isAuthenticated}= require('../helpers/auth')

Cada ruta la asegurare asi:

router.get('/notes',isAuthenticated,async (req,res)=>{

Agrego una variable global para darle una bienvenida al usuario cuando se loggee, para ello creo otra variable global. Luego lo uso en all-notes.hbs por ejemplo para al momento de entrar y no hayan notas mostrar su nombre.

Pero la variable global también servirá para, al ya estar loggeado dejar de mostrar el SignIn y SignUp. Para ello voy al partial/navigation.hbs pero aquí solo se oculta de manera visual con un #if user, en caso de asegurar ya tenemos isAuthenticated