# Road Side Coder MERN youtube

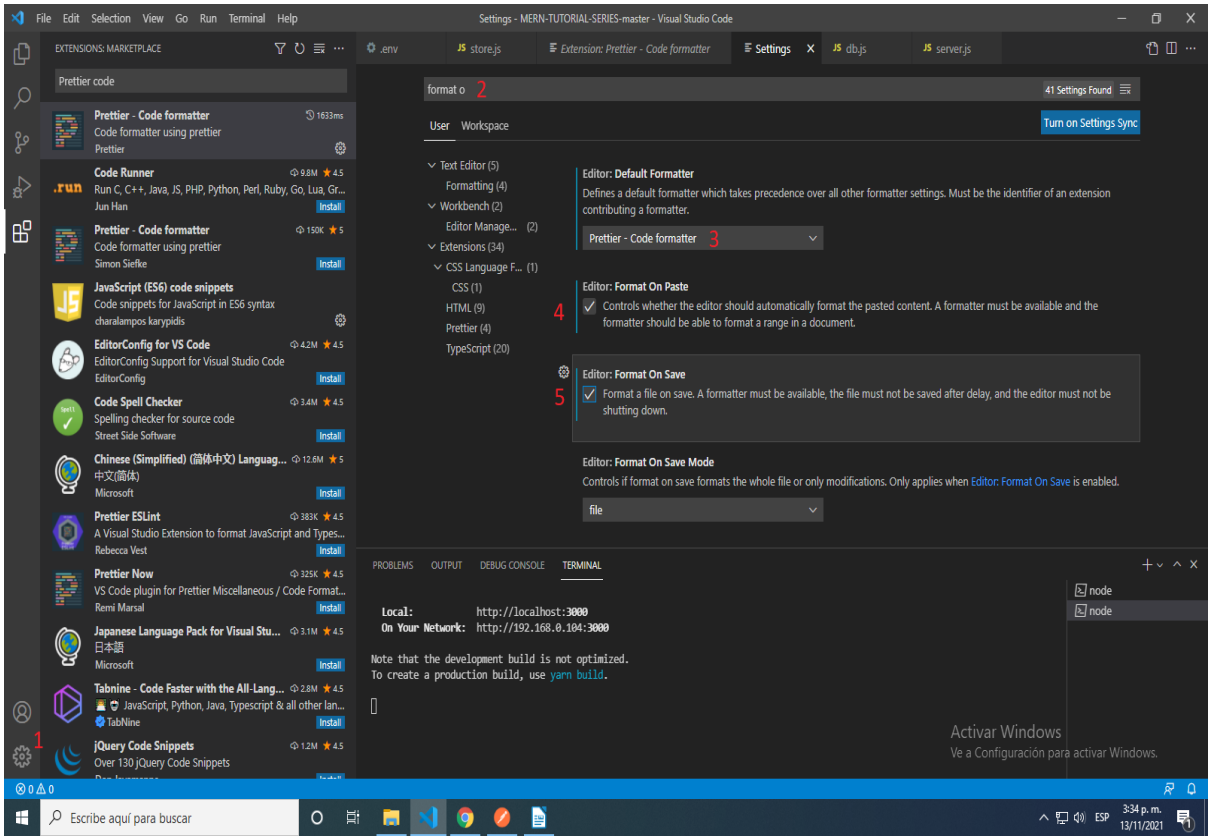
.env

```
PORT=5000
MONGO_URI=mongodb+srv://
ricardo:kessler200@cluster0.dcjlt.mongodb.net/road?
retryWrites=true&w=majority
NODE_ENV=development
JWT_SECRET=mysecret
```

**Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.**

1. VSCODE :
Librerías que pueden ahorrate mucho tiempo:

*Auto Rename Tag(Cambia las <Tag></Tag> automáticamente al editar)
* Bracket Pair Colorized 2(Colorea las {} , () para saber donde terminan y empiezan)
*ES7 React....Snippets(Crea BoilerPlates)
*JavaScript ES6 Snippets
*Material Icon Theme
*Prettier code formatter(En el minuto 6 explica como configurarlo correctamente) 1. Botòn Configuraciòn 2. Buscar format o, 3. prettier-code formatter 4. On paste, 5.On save.



2. VIDEO 4 → 5:00 → va a mejorar la llamada al server.js

*Api endpoint es donde el router que contiene el dato de la api es servido

*Ejemplo usando dummy data(dato de archivo estático)

```
const notes = require("./data/notes");

app.get("/api/notes", (req, res) => {
      res.json(notes);
      });
```

Quedé en el minuto 10:00

*10:50 Deatil Api View

`req.params` **Nos va a dar todos los parametros que están en nuestra id**

```
app.get("/api/notes/:id", (req, res) => {
const note = notes.find((n) => n._id === req.params.id);
console.log(req.params);
});
```

* 13:50 → .env file (Explicación) Secrete information about our aplication

* 14: 32 → Hay que instalar **npm i dotenv**

* Edit . PORT = 5000

Quedé 15:02 **Instalación de nodemon**

Para Guardar cambios automáticamente sin necesidad de reiniciar el server

VIDEO 5 FrontEnd

10:30 Como subir todo a github → “Importante hay varios detalles por que son dos carpetas”

Vuelve a notezipper-mern

TUVE QUE BORRAR EL **.GIT** DEL FRONTEND POR QUE TENÍA CONFLICTOS CON EL **.GIT** DE EL DIRECTORIO ROOT

Hay que tener en cuenta por ejemplo el archivo de .env

El archivo **.gitignore** que tienes en src debes pasarlo al directorio principal al lado de **.env**

y Editarlo asi quedó, los cambios están en azul…

```
# See https://help.github.com/articles/ignoring-files/ for more about ignoring files.

# dependencies
/node_modules
node_modules/
frontend/node_modules/
/.pnp
.pnp.js

# testing
/coverage

# production
/build

# misc
.DS_Store
.env
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

* Hacer git init en el **root directory**

## VIDEO 6  LANDING PAGE AND NAVBAR

16:06 Quedé → Landing Page

VIDEO 7  REACT ROUTER

Quedé 17 : 35

VIDEO 8 CONNECTING BACKEND WITH FRONTEND

Al principio no podía por que faltaba la carpeta de notes que estaba en frontend y la necesitaba en backend , el no me lo recordó o debe ser que no la llamé apropiadamente desde el frontend. **REVISA ESO**

**11:00 → Utiliza "concurrently" para correr todo al mismo timpo (back and frontend)**

# npm run dev (Para correr Ambos)

## 5:00 VIDEO 9

No se si sirve , puse mydb (que ya tiene una tabla o collection posts) como data base, hay que probar, o tal vez necesites crear una nueva

**Creó un archivo para la conexión llamado db.js en la carpeta config**

```
const mongoose = require('mongoose')

const connectDB = async ()=>{
try {
const conn = await mongoose.connect(process.env.MONGO_URI)
console.log(`MongoDB Conectado: ${conn.connection.host}`)
} catch (error) {
console.error(`Error: ${error.message}`)
process.exit()
}
}

module.exports = connectDB;
```

Al que después llamo por:

```
const express = require("express");
const notes = require("./data/notes");
const dotenv = require("dotenv");
const connectDB = require("./config/db")

const app = express();
dotenv.config();
connectDB();
```

**VIDEO 10**

User Authentication JWT

* Si solo necesitas correr el servidor : **node backend/server.js**

* Instalé **POSTMAN**

*** express.Router**

**Router** para **crear manejadores de rutas** montables y modulares. Una instancia **Router** es un sistema de middleware y direccionamiento completo.

*** Creación de una variable Url – 7:00**

**¿Qué hace App use?**

use (middleware) se llama cada vez que se envía una solicitud al servidor. app.use(), que se utiliza para montar la función de middleware o para montar en una ruta específica, la función de middleware se ejecuta cuando la ruta de base coincide.

**Express** es una infraestructura web de direccionamiento y middleware que tiene una funcionalidad mínima propia: una aplicación **Express** es fundamentalmente una serie de llamadas a funciones de middleware.

* Enlace el middleware de nivel de aplicación a una instancia del objeto de aplicación utilizando las funciones app.use() y app.METHOD(), donde METHOD es el método HTTP de la solicitud

que maneja la función de middleware (por ejemplo, GET, PUT o POST) en minúsculas.

Este ejemplo muestra una función de middleware sin ninguna vía de acceso de montaje. La función se ejecuta cada vez que la aplicación recibe una solicitud.

```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

```
connectDB();
app.use(express.json()) //para analizar la aplicación / json //
para aceptar datos del usuario
```

Cuando se habla de `express.json()` y `express.urlencoded()` piense específicamente en POST peticiones (es decir, el objeto de solicitud .post) y Solicitudes PUT (es decir, el objeto de solicitud .put )

1. NO NECESITA `express.json()` y `express.urlencoded()` para las solicitudes GET o DELETE.

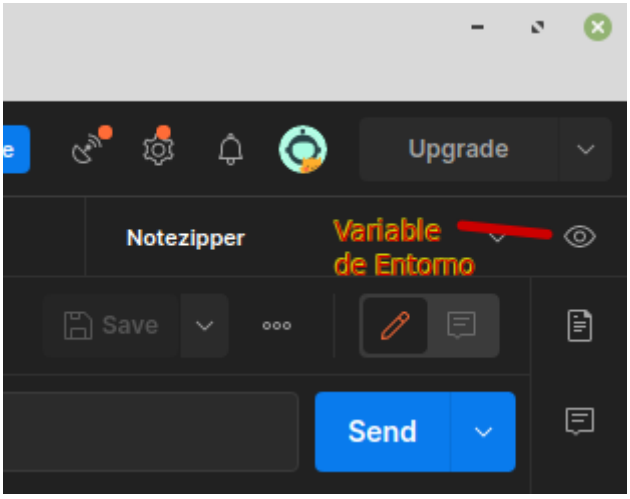
2. NECESITAS `express.json()` y `express.urlencoded()` para POST y PUT solicitudes, porque en ambas solicitudes estás **enviando datos** (en la forma de algún objeto de datos) al servidor y le está pidiendo al servidor que acepte o almacene esos datos (objeto), que está encerrado en el cuerpo (es decir, `req.body`) de esa solicitud (POST o PUT)

3. Express le proporciona middleware para tratar con los datos (objetos) (entrantes) en el cuerpo de la solicitud.

   a. `express.json()` es un método incorporado en express para reconocer el objeto de solicitud entrante como **objeto JSON**. Este método se llama como middleware en su aplicación usando el código: `app.use(express.json())`;

si. `express.urlencoded()` es un método incorporado en express para reconocer el objeto de solicitud entrante como **cadenas o matrices**. Este método se llama como middleware en su aplicación usando el código: `app.use(express.urlencoded());`

* SET a {{URL}} Variable en postman – 7:10




Add Enviroment Variable no, global

* Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar **next()** para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.

* Como Encriptar el password para que no sea visible en **mongo-cloud** – 18:17

```
// will encrypt password everytime its saved
// prev significa que va a hacerlo antes de introducir los datos

userSchema.pre("save", async function (next) {
if (!this.isModified("password")) {
next();
}
const salt = await bcrypt.genSalt(10);
this.password = await bcrypt.hash(this.password, salt);
});
```

**Debes crear un nuevo usuario , simplemente hacer un pequeño cambio en email para probar.**

Quedé en 23:35

En el minuto 29 hace un resumen explicativo

JWT – 30:00

Llevar el token al frontend

* React Login 33:33
* Es hora de llamar a la API 41:31

Creacion de Componente de Error y Loading 45:02

Funcionabilidad de Register –  52:34,

Quedé en 58:14, Cloudinary

Quedé en 59:20, Cloudinary WOOWWWW

REPASO JWT -10

TAREAS

1. Si

```
  if (userExists) {
     res.status(400);
     throw new Error("User Already Exist(email)");
}
```

**Devuelve esto:**


```
{
"message": "User Already Exist(email)",
"stack": "Error: User Already Exist(email)\n at
/home/ricardo/React/MERN-TUTORIAL-SERIES/backend/controllers/u
serControllers.js:12:11\n at processTicksAndRejections
(node:internal/process/task_queues:96:5)"
```

```
}
```

**Podria consumir esa respuesta con fecth o axios ?**
**https://www.youtube.com/watch?v=lkpXNa2ZIjY BLUUWEB**

```
res.json({
  error: null, // tal vez pasar una var bandera como esta y que react la valide y nos devuelva un mensaje.
  data: {
  title: "mi ruta protegida",
 user: req.user,
},
```

2.

En el proyecto de refresh token llama a usuario de manera estatica, pero en el de bluuweb. Creo q lo llama así:

auth.js

```
const User = require("../models/User");
```

REPASO JWT -10

```
router.post("/login", async (req, res) => {

      const { error } = schemaLogin.validate(req.body);
      if (error) return res.status(400).json({ error: error.details[0].message });

      const user = await User.findOne({ email: req.body.email });
      if (!user) return res.status(400).json({ error: "Usuario no encontrado" });

      res.json({
            error: null,
            data: "exito bienvenido",
            user, ← Esto es una teoría, debo probar
      });
```

https://www.youtube.com/watch?v=lkpXNa2ZIjY **BLUUWEB 4:40**

**3.** const **asyncHandler** = require("**express-async-handler**"); ← Woww esto maneja más rápido async y await automaticamente. ← Minuto 14

aunque igual usa await, debes estudiar eso

4. ( 23:22 ) **Middleware is a function with access to the request object (req), the response object (res), and the next middleware in the application's request-response cycle, commonly denoted by a variable named next.**

**Middleware can:**

- **Execute any code.**
- **Make changes to the request and the response objects.**
- **End the request-response cycle.**
- **Call the next middleware in the stack.**

**5. Register**

*Comparar passwords(Si está seguro de su nuevo password)- **55:17**

* **Usando Objetos FormData – 1:02:14**