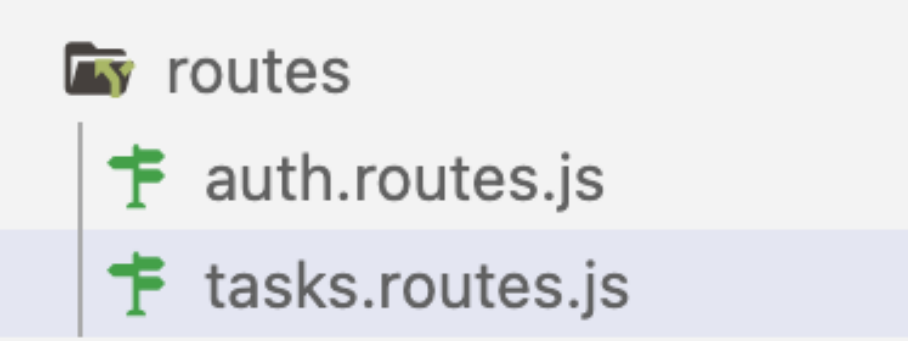
Configurando MongoDB Atlas – mongoose – express – morgan – bycryptjs - jsonwebtoken.

Parte – 5
Operaciones CRUD

En esta parte del documento vamos a trabajar sobre las diferentes operaciones que podemos hacer sobre nuestra base de datos de mongodb. Estas operaciones son las básicas y son conocidas como operaciones CRUD

- **C ->** Create: crear una base de datos.
- **R ->** Read: leer de una base de datos.
- **U ->** Upload: Actualizar la base de datos.
- **D ->** Delete: Borrar de la base de datos.

Lo primero que vamos a crear es dentro de la carpeta rutas, un fichero denominado "task.routes.js" en los que vamos a definir las diferentes rutas para lanzar las diferentes operaciones sobre las bases de datos.



El contenido del fichero "task.routes.js" es el que se muestra a continuación:

```
import { Router } from 'express';
import { authRequired } from '../middlewares/validateToken.js';
import { getTasks, getTask, createTask, updateTask, deleteTask } from '../controllers/tasks.controller.js';
import { validateSchema } from '../middlewares/validator.middleware.js';
import { createTaskSchema } from '../schemas/task.schema.js';

const router = new Router();

router.get('/tasks', authRequired, getTasks);  //coger tareas
router.get('/tasks/:id', authRequired, getTask); //coger una tarea de un usuario.
router.post('/tasks', authRequired, validateSchema(createTaskSchema), createTask);  //añadir tareas
router.delete('/tasks/:id', authRequired, deleteTask);  //eliminar tareas
router.put('/tasks/:id', authRequired, updateTask); //actualizar las tareas

export default router;
```


IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

Importamos la función **Router de express** para poder generar las diferentes rutas que configuramos en nuestro backend.

Importamos nuestra función **authRequired** desde nuestra carpeta de seguridad en el fichero **validateToken.js** y que hemos visto en los apuntes anteriores.

```
import jwt from 'jsonwebtoken';
import { TOKEN_SECRET } from '../config.js';

export const authRequired = (req, res, next) => {
    const { token } = req.cookies;

    if (!token)
        return res.status(401).json({ mesage: "No token, autenticación rechazada" });


jwt.verify(token, TOKEN_SECRET, (err, user) => {
    if(err) return res.status(403).json({mesage: "Token inválido"});

    req.user = user;

    next();
});

}
```

Importamos las funciones que vamos a asociar con cada una de las rutas. Es importante que os acordéis de que cuando hablamos de rutas, tenemos funciones intermedias que son las que si se ejecutan sin ningún problema, nos devolverán un next() para lanzar la siguiente función que tenemos en la ruta. Siguiendo esto, es importante el orden en el que ponemos estas funciones.

Las dos últimas importaciones que hacemos son una validación que hacemos de los datos antes de enviarlos al servidor. Para ello utilizamos la función "createTaskSchema()". Para poder utilizar este tipo de validación, utilizamos un framework que es "ZOD". Se pueden hacer muchas validaciones sobre diferentes tipos de datos. Tenéis toda la información para trabajar con el en la web http://www.zod.dev.

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

El contenido del fichero createTaskSchema() es el siguiente:

```
import { z } from 'zod';

export const createTaskSchema = z.object({
    title: z
        .string({
            required_error: "Debes añadir un título",
        }),
    description: z
        .string({
            required_error: "La descripción debe de ser un string",
        }),
    date: z
        .string().datetime().optional(),
});
```

Observar que importo 'zod' y lo que hago es poner condiciones en un esquema que le vamos a mandar al backend. En este caso le digo que los campos van a ser de tipo string, que son requeridos o que el campo de la fecha, que es de un tipo especial de string -> .string().datetime() es opcional.

La función de "validateSchema()", nos queda de la siguiente manera:

```
1
2   export const validateSchema = (schema) => (req, res, next) => {
3       try {
4           schema.parse(req.body);
5           next();
6       } catch (error) {
7           return res.status(400).json({ error: error.errors.map(error => error.message) });
8       }
9   }
```

Es importante que observéis que a la hora de crear las diferentes rutas, estamos poniendo diferentes protocolos como el POST, GET, DELETE y el PUT, según queramos añadir información a la base de datos o recoger datos de la base de datos.

Fijaros también que cuando añadimos a nuestra base de datos una colección de datos referente a las tareas, nos va a crear una nueva carpeta que se denominará "tasks".

El fichero que contiene las funciones que van a generar las operaciones CRUD sobre nuestra base de datos, es el "tasks.controller.js" y su contenido es el siguiente:

IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891

```
import Task from '../models/task.model.js';


export const getTasks = async (req, res) => {
    const tasks = await Task.find({ user: req.user.id }).populate("user")

    res.json(tasks)

};

export const createTask = async (req, res) => {
    const { title, description, date } = req.body;
    const newTask = new Task({
        title,
        description,
        date,
        user: req.user.id,
    })

    const saveTask = await newTask.save();
    res.json(saveTask);
};

export const getTask = async (req, res) => {
    const task = await Task.findById(req.params.id).populate('user');

    if (!task)
        return res.status(404).json({ message: "Tareas no encontradas" });
    res.json(task);
};

export const deleteTask = async (req, res) => {
    const task = await Task.findByIdAndDelete(req.params.id);

    if (!task)
        return res.status(404).json({ message: "No se ha eliminado nada" });
    res.sendStatus(204);
};
 export const updateTask = async (req, res) => {
     const task = await Task.findByIdAndUpdate(req.params.id, req.body, {
         new: true,
     });

     if (!task)
         return res.status(404).json({ message: "No se ha eliminado nada" });
     res.json(task);
 };
```


IES VLLABLANCA
C/Villablanca, 79 - 28032 Madrid
Teléfono 917 752 214 – Fax 917 766 577
e-mail ies.villablanca.madrid@educa.madrid.org
C.C.: 28039891