



Curso complementario Desarrollo de Back-end con Node.js - MongoDB

ING - DIEGO CASALLAS

22810019-1



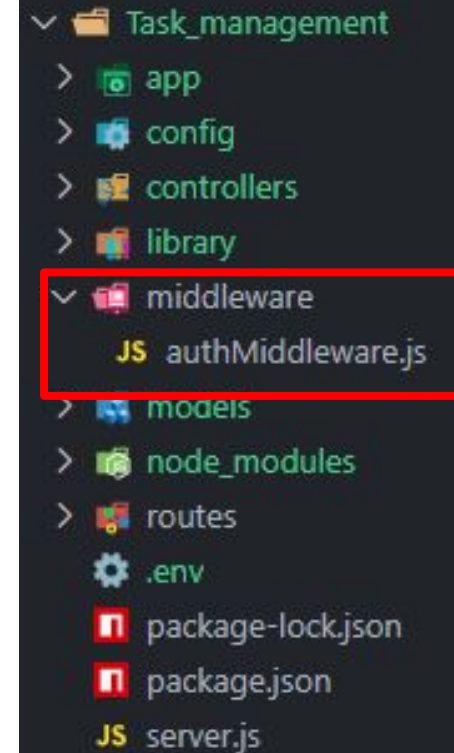
www.sena.edu.co

Proyecto Paso a Paso



- middleware -> **authMiddleware.js**
 - Validar token

```
Task_management > middleware > JS authMiddleware.js > [?] verifyToken
1  import jwt from "jsonwebtoken";
2  import dotenv from "dotenv";
3
4  dotenv.config();
5
6  // Middleware to verify the token
7  export const verifyToken = (req, res, next) => {
8    const token = req.header("Authorization");
9    if (!token) return res.status(401).json({ error: "Access denied" });
10
11    try {
12      const verified = jwt.verify(token.replace("Bearer ", ""), process.env.JWT_SECRET);
13      req.user = verified;
14      //See data token encrypted
15      //console.log(verified);
16      next();
17    } catch (err) {
18      res.status(400).json({ error: "Invalid Token" });
19    }
20  };
```

A file explorer view of the project structure. The 'middleware' folder is highlighted with a red box, and the 'authMiddleware.js' file is listed below it. The project structure includes: Task_management (expanded), app, config, controllers, library, middleware (expanded), models, node_modules, routes, .env, package-lock.json, package.json, and server.js.

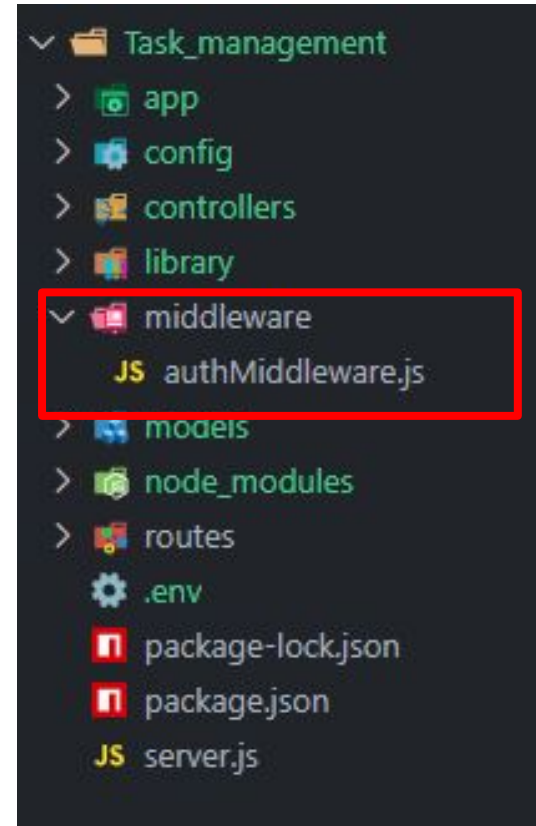
```
Task_management
├── app
├── config
├── controllers
├── library
├── middleware
│   └── JS authMiddleware.js
├── models
├── node_modules
├── routes
├── .env
├── package-lock.json
├── package.json
└── JS server.js
```

Proyecto Paso a Paso



- **Implementar en las rutas**
 - llamar el middleware a la ruta que se va a implementar.

```
Task_management > routes > JS auth.routes.js > [🔗] default
1  import { Router } from "express";
2  import AuthController from '../controllers/auth_controller.js';
3  import {verifyToken} from '../middleware/authMiddleware.js';
4  const router = Router();
5
6  // Public route
7  router.post('/auth/register',verifyToken, AuthController.register);
8  router.post('/auth/login', AuthController.login);
9
10 export default router;
```



Proyecto Paso a Paso



● Probar

HTTP TASK-MONGO-NODE / auth / Register

POST http://localhost:3000/api/auth/register

Params Authorization Headers (10) Body Scripts Tests Settings

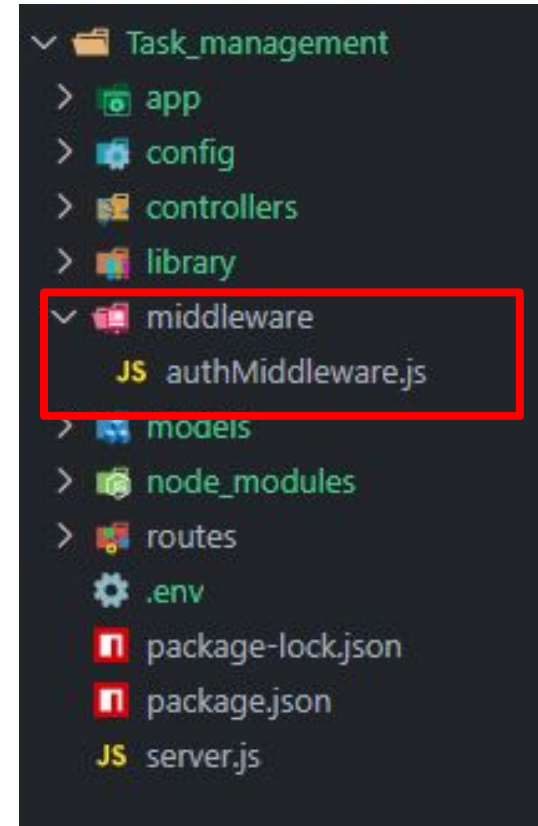
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "username": "diehercasvan1",
3   "email": "diehercasvan1@gmail.com",
4   "password": "Die80@59877*"
5 }
```

Body Cookies (1) Headers (7) Test Results 401 Unauthorized

{ } JSON Preview Visualize

```
1 {
2   "error": "Access denied"
3 }
```



Proyecto Paso a Paso



- **CRUD de usuario**

- Usamos el mismo modelo de **user**
- Crear el controlador
- Crear la ruta
 - Implementar middleware
- **Realizar pruebas**

A screenshot of a file explorer window showing the project structure. The 'controllers' folder is expanded, and 'user.controller.js' is highlighted with a red rectangle. Other files and folders visible include 'Task_management', 'app', 'config', 'library', 'middleware', 'models', 'node_modules', 'routes', '.env', 'package-lock.json', 'package.json', and 'server.js'.

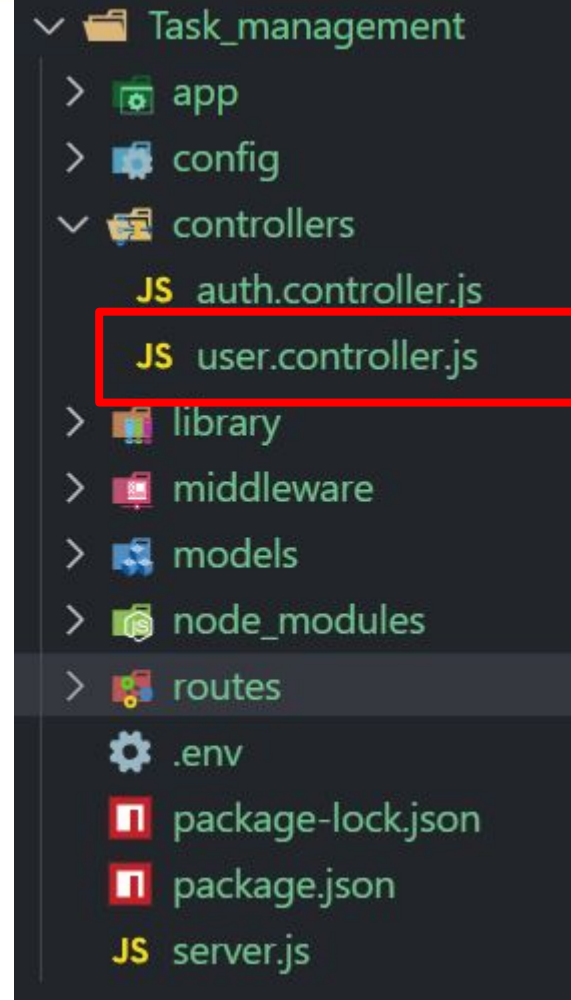
```
Task_management
├── app
├── config
├── controllers
│   ├── auth.controller.js
│   └── user.controller.js
├── library
├── middleware
├── models
├── node_modules
├── routes
├── .env
├── package-lock.json
├── package.json
└── server.js
```


Proyecto Paso a Paso



- controllers -> user.controller.js

```
Task_management > controllers > JS user.controller.js > UserController
1  import UserModel from '../models/user.model.js'; // Import the UserModel
2  import dotenv from 'dotenv';
3
4  dotenv.config();
5  class UserController {
6    // User registration
7    async addUser(req, res) {
8      const passwordRegex = /^(?=.*\d)(?=.*[\u0021-\u002b\u003c-\u0040])(?=.*[A-Z])(?=.*[a-z])\S{8,16}$/;
9      try {
10       const { username, email, password } = req.body;
11       if (!username || !email || !password) {
12         return res.status(400).json({ error: 'All data is mandatory for entry' });
13       }
14       if (!passwordRegex.test(password)) {
15         return res.status(400).json({ error: 'The password must be between 8 and 16 characters long, with at least one digit, at least one lowercase letter, at least one uppercase letter, and at least one non-alphanumeric character.' });
16       }
17       const existingUserModel = await UserModel.findOne({ email });
18       if (existingUserModel) {
19         return res.status(400).json({ error: 'The user already exists' });
20       }
21       const newUserModel = new UserModel({ username, email, password });
22       await newUserModel.save();
23       return res.status(201).json({ message: 'User successfully registered' });
24     } catch (err) {
25       res.status(400).json({ error: err.message });
26     }
27   };
28 }
```

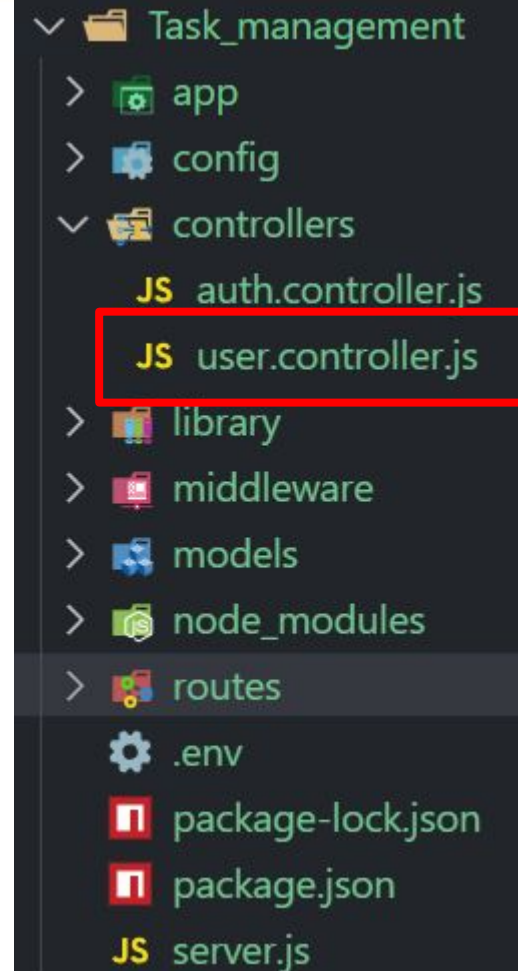
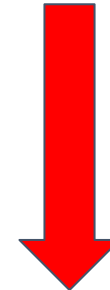


Proyecto Paso a Paso



- **controllers -> user.controller.js**

```
29 // Users list
30 async show(req, res) {
31   try {
32     const userModel = await UserModel.find();
33     if (!userModel) throw new Error('User not found');
34     return res.status(200).json({ data: userModel });
35   } catch (err) {
36     res.status(400).json({ error: err.message });
37   }
38 };
39
```



Proyecto Paso a Paso



- **controllers -> user.controller.js**

```
40 // Find user by ID
41 async findById(req, res) {
42   try {
43     const id = req.params.id;
44     if (!id) {
45       return res.status(400).json({ error: 'User Id is required' });
46     }
47     const userModel = await UserModel.findOne({ _id: id });
48     if (!userModel) throw new Error('User not found');
49     return res.status(200).json({ data: userModel });
50   } catch (err) {
51     res.status(400).json({ error: err.message });
52   }
53 };
```



Task_management

- > app
- > config
- ▼ controllers
 - JS auth.controller.js
 - JS user.controller.js**
- > library
- > middleware
- > models
- > node_modules
- > routes

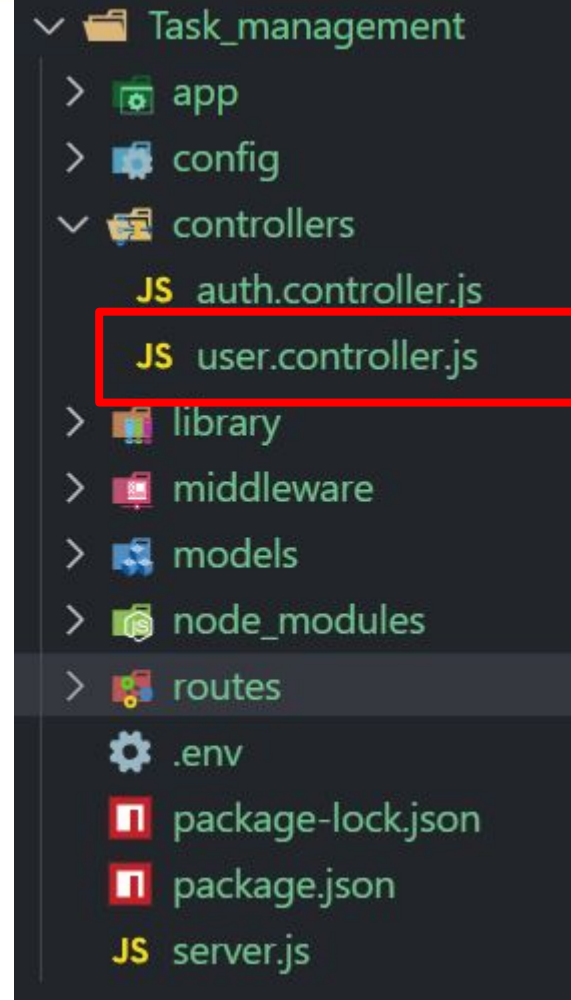
.env
package-lock.json
package.json
JS server.js

Proyecto Paso a Paso



- controllers -> user.controller.js

```
55 // User update
56 async update(req, res) {
57   const passwordRegex = /^(?=.*\d)(?=.*[\u0021-\u002b\u003c-\u0040])(?=.*[A-Z])(?=.*[a-z])\S{8,16}$/;
58   try {
59     const { username, email, password } = req.body;
60     if (!username || !email || !password) {
61       return res.status(400).json({ error: 'All data is mandatory for entry' });
62     }
63     if (!passwordRegex.test(password)) {
64       return res.status(400).json({ error: 'The password must be between 8 and 16 characters long, with at least one digit, at least one lowercase letter, at least one uppercase letter, and at least one non-alphanumeric character.' });
65     }
66     const existingUserModel = await UserModel.findOne({ email });
67     if (!existingUserModel) {
68       return res.status(400).json({ error: 'User not found' });
69     }
70     const updateUser = await UserModel.findOneAndUpdate(
71       { _id: req.params.id }, // Filtro
72       { username, password }, // Datos nuevos
73       { new: true } // Devuelve el documento actualizado
74     );
75     if (!updateUser) {
76       return res.status(404).json({ error: 'User not updated' });
77     }
78     res.status(200).json(updateUser);
79   } catch (error) {
80     res.status(400).json({ error: error.message });
81   }
82 };
```

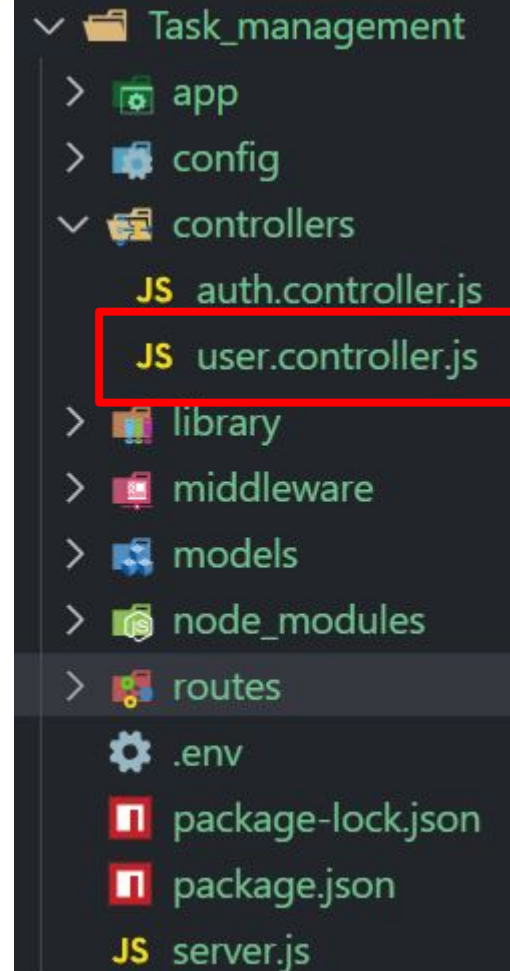


Proyecto Paso a Paso



- **controllers -> user.controller.js**

```
84 // User deletion
85 async delete(req, res) {
86   try {
87     const deletedUser = await UserModel.findOneAndDelete({
88       _id: req.params.id
89     });
90     if (!deletedUser) {
91       return res.status(404).json({ error: 'User not updated' });
92     }
93     res.status(200).json({ message: 'Deleted successfully' });
94   } catch (error) {
95     res.status(500).json({ error: 'Error deleting user' });
96   }
97 };
98
99 }
100 export default new UserController();
```

A file explorer view of a project directory. The 'controllers' folder is expanded, showing 'auth.controller.js' and 'user.controller.js'. 'user.controller.js' is highlighted with a red box. A large red arrow points from the code editor on the left towards the file explorer on the right.

- Task_management
 - app
 - config
 - controllers
 - JS auth.controller.js
 - JS user.controller.js**
 - library
 - middleware
 - models
 - node_modules
 - routes
- .env
- package-lock.json
- package.json
- JS server.js

Proyecto Paso a Paso



- **routes -> user.routes.js**

```
Task_management > routes > JS user.routes.js > ...
```

```
1  import { Router } from "express";
2  import UserController from '../controllers/user.controller.js';
3  import {verifyToken} from '../middleware/authMiddleware.js';
4
5  const router = Router();
6  const name="/user";
7  // Verify token middleware
8  router.use(verifyToken);
9  // Route for user registration and list
10 router.route(name)
11   .post(UserController.addUser)
12   .get(UserController.show);
13 //Route for user by ID
14 router.route(`${name}/:id`)
15   .get(UserController.findById)
16   .put(UserController.update)
17   .delete(UserController.delete);
18
19 export default router;
```

CONTROLLER

```
Task_management
├── app
├── config
├── controllers
├── library
├── middleware
├── models
├── node_modules
├── routes
│   ├── auth.routes.js
│   └── user.routes.js
├── .env
├── package-lock.json
├── package.json
└── server.js
```


Proyecto Paso a Paso

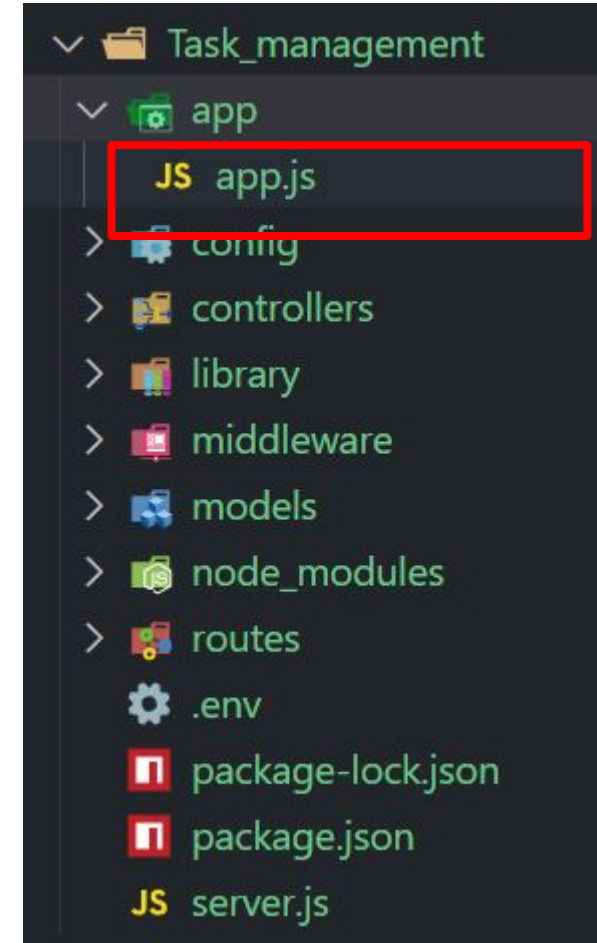


- **app -> app.js**

ROUTER

ADD ROUTER

```
Task_management > app > JS app.js > ...
1  import express from 'express';
2  import { connectDB } from '../config/db/connection.js';
3  // Import routes
4  import authRouter from '../routes/auth.routes.js';
5  import userRouter from '../routes/user.routes.js';
6
7  const app = express();
8  app.use(express.json());
9  app.use(express.urlencoded({ extended: true }));
10
11 // Connect to MongoDB
12 connectDB();// Call the connection function
13
14 // Use routes
15 app.use('/api', authRouter);
16 app.use('/api', userRouter);
17
18 app.use((req, res, next) => {
19   res.status(404).json({
20     message: 'Endpoint losses'
21   });
22 });
23
24 export default app;
```



Proyecto Paso a Paso



● Pruebas

GET http://localhost:3000/api/user

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Auth Type Bearer Token

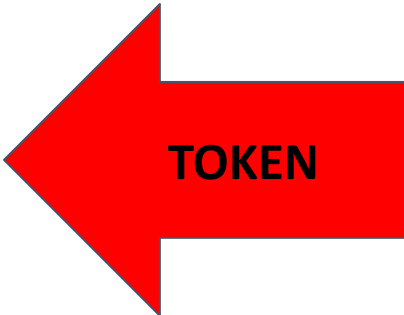
Token

The authorization header will be automatically generated when

Body Cookies (1) Headers (7) Test Results

200 OK 18 ms 595 B

```
1 {
2   "data": [
3     {
4       "_id": "68759c73b17af6b2730cddd4",
5       "username": "diehercasvan1",
6       "email": "diehercasvan1@gmail.com",
7       "password": "$2b$10$JeVkoz.x7A.c.ot6BYTtz.QKrsZP0gt3Gu1Vh6pB0S1xbsnv94Um6",
8       "_v": 0
9     },
10    {
11      "_id": "6875cfc7fb0f3af1da9f08dd",
12      "username": "diehercas",
13      "email": "diehercas@gmail.com",
14      "password": "$2b$10$JeVkoz.x7A.c.ot6BYTtz.QKrsZP0gt3Gu1Vh6pB0S1xbsnv94Um6",
15      "_v": 0
16    }
17  ]
18 }
```



TASK-MONGO-NODE

auth

user

GET show

GET showId

POST add

PUT update

DEL delete

Método	Ruta	Controlador	Modelo
GET	http://localhost:3000/api/user	User	User

Proyecto Paso a Paso



- **Pruebas**

GET

http://localhost:3000/api/user/68759c73b17af6b2730cddd4

Send

Params

Authorization

Headers (9)

Body

Scripts

Tests

Settings

Cookies

Auth Type

Bearer Token

Token

The authorization header will be automatically generated when

Body

Cookies (1)

Headers (7)

Test Results

200 OK

12 ms

422 B

Save Response

JSON

Preview

Visualize

1 {

2 "data": {

3 "id": "68759c73b17af6b2730cddd4",

4 "username": "diehercasvan1",

5 "email": "diehercasvan1@gmail.com",

6 "password": "\$2b\$10\$JeVkoz.x7A.c.ot6BYTtz.QKrsZP0gt3Gu1Vh6pB0S1xbsnv94Um6",

7 "__v": 0

8 }

9 }

← ID

← TOKEN

← RESULTADO

TASK-MONGO-NODE

auth

user

GET show

GET showId

POST add

PUT update

DEL delete

Método	Ruta	Controlador	Modelo
GET	http://localhost:3000/api/user:id	User	User

Proyecto Paso a Paso



● Pruebas

POST

http://localhost:3000/api/user

Params

Authorization

Headers (11)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1 {

2 | "username": "newuser", "email": "newuser@gmail.com", "password": "Newuser80859867"

3 }

Body

Cookies (1)

Headers (7)

Test Results

201 Created

175 ms

282 B

{}

JSON

Preview

Visualize

1 {

2 | "message": "User successfully registered"

3 }

TOKEN

DATOS

RESULTADO

TASK-MONGO-NODE

auth

user

show

showId

add

update

delete

Método	Ruta	Controlador	Modelo
POST	http://localhost:3000/api/user	User	User

Proyecto Paso a Paso



● Pruebas

Diagram illustrating the API test setup and response:

- TOKEN** (Red arrow pointing left)
- ID** (Cyan arrow pointing left)
- DATOS** (Blue arrow pointing left)
- RESULTADO** (Green arrow pointing left)

API Test Configuration (PUT):

- URL: `http://localhost:3000/api/user/68768746acdd080a7c6e3b6f`
- Params: Authorization (highlighted in red)
- Headers (11)
- Body (highlighted in blue):

```
1 {
2   "username": "NewUse", "email": "newuser@gmail.com", "password": "Newuser80800867*"
3 }
```
- Scripts
- Tests
- Settings

Response (200 OK):

```
1 {
2   "_id": "68768746acdd080a7c6e3b6f",
3   "username": "NewUse",
4   "email": "newuser@gmail.com",
5   "password": "Newuser80800867*",
6   "__v": 0
7 }
```

API Explorer (TASK-MONGO-NODE):

- auth
 - > folder icon
- user
 - GET show
 - GET showId
 - POST add
 - PUT update** (highlighted in red)
 - DEL delete

Método	Ruta	Controlador	Modelo
PUT	<code>http://localhost:3000/api/user:id</code>	User	User

Proyecto Paso a Paso



- **Pruebas**

DELETE

http://localhost:3000/api/user/68768746acdd080a7c6e3b6f

Params

Authorization

Headers (9)

Body

Scripts

Tests

Settings

Auth Type

Bearer Token

Token

.....

Body

Cookies (1)

Headers (7)

Test Results

200 OK

14 ms

269 B

Save Response

JSON

Preview

Visualize

1 {

2 "message": "Deleted successfully"

3 }

TASK-MONGO-NODE

auth

user

GET show

GET showId

POST add

PUT update

DEL delete

ID

TOKEN

RESULTADO

Método	Ruta	Controlador	Modelo
DELETE	http://localhost:3000/api/user:id	User	User

Actividad



- **Crear el modelo de datos para almacenar tareas**
 - **Con los siguientes campos:**
 - **Información básica:**
 - `_id`: Identificador único de MongoDB
 - `título, descripción`: Detalles de la tarea
 - `tipo`: Categoría de la tarea (Desarrollo, Diseño, Reunión, etc.)
 - `prioridad`: Nivel de prioridad (Alta, Media, Baja)
 - `color`: Código hexadecimal para identificación visual
 - `estado`: Estado actual (Pendiente, En progreso, Completada, etc.)
 - **Fechas:**
 - `fechaInicio, fechaFin`: Rango de tiempo para la tarea
 - `fechaCreacion`: Cuando se creó el registro
 - `ultimaModificacion`: Última actualización
 - **Personas involucradas:**
 - `correoNotificacion`: Email para notificaciones
 - `encargados`: Array con los responsables
 - `creadoPor`: Quién creó la tarea
 - **Archivos adjuntos:**
 - `archivos`: Array con documentos relacionados
 - **Otros:**
 - `comentarios`: Discusiones sobre la tarea
 - `etiquetas`: Palabras clave
 - `proyectoId`: Proyecto al que pertenece

Documentación Oficial

<https://mongoosejs.com/docs/guide.html>

- **Crear modelo**
- **Crear el controlador**
- **Crear las rutas**
 - **validar token**
- **Realizar las pruebas (Postman)**
 - **Crear documentación**

Mongoose JS con Node.js



- **¿Qué es Mongoose?**
 - ODM (Object Document Mapper):
 - Permite interactuar con MongoDB usando objetos y esquemas en JavaScript.
- **Ventajas:**
 - Validación de datos.
 - Middlewares (hooks).
 - Consultas más intuitivas.
- **Diferencia con el Driver Nativo:**
 - Mongoose añade estructura, mientras que el driver trabaja con MongoDB directamente.

Mongoose JS con Node.js



- **Instalación y Configuración**

```
npm install mongoose
```

- **Conexión a MongoDB:**

```
const mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost:27017/miDB', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
});
```

Mongoose JS con Node.js



- **Esquemas (Schemas) y Modelos**

```
const userSchema = new mongoose.Schema({  
  name: { type: String, required: true },  
  age: { type: Number, min: 18 },  
  email: { type: String, unique: true }  
});
```

- **Modelo: Representación de una colección en la DB.**

```
const User = mongoose.model('User', userSchema);
```

Mongoose JS con Node.js



- **Operaciones CRUD**

- **Create**

- ```
const newUser = new User({ name: "Ana", age: 25 });
newUser.save();
```

- **Read:**

- ```
User.find({ age: { $gte: 18 } });
```

- **Update:**

- ```
User.updateOne({ name: "Ana" }, { age: 26 });
```

- **Delete:**

- ```
User.deleteOne({ name: "Ana" });
```

Mongoose JS con Node.js



- **Middlewares (Hooks)**

- **Usos comunes:**

- Encriptar contraseñas antes de guardar.
 - Validaciones personalizadas.

```
userSchema.pre('save', function(next) {  
  console.log("Guardando usuario...");  
  next();  
});
```


Mongoose JS con Node.js



- **Populate (Relaciones entre Colecciones)**

- **con Referencias:**

```
const postSchema = new mongoose.Schema({  
  title: String,  
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }  
});
```

- **con Populate:**

```
Post.findOne({ title: "Hola Mundo" }).populate('author');
```

Mongoose JS con Node.js



- Ejemplo

```
import mongoose from 'mongoose';
const { Schema } = mongoose;

const blogSchema = new Schema({
  title: String, // String is shorthand for {type: String}
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

Mongoose JS con Node.js



- **Recursos Adicionales**

- Documentación Oficial: mongoosejs.com
- Repositorio GitHub: github.com/Automattic/mongoose
- Curso Recomendado: [MongoDB University](https://mongodb.university)



G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



@SENAComunica

www.sena.edu.co