Open in app ↗ ...up Sign in

Medium Search

Building a Secure Node.js API: Managing Users, Roles, and Tasks with JWT

Alberto R. · Follow
20 min read · 4 days ago

Listen Share

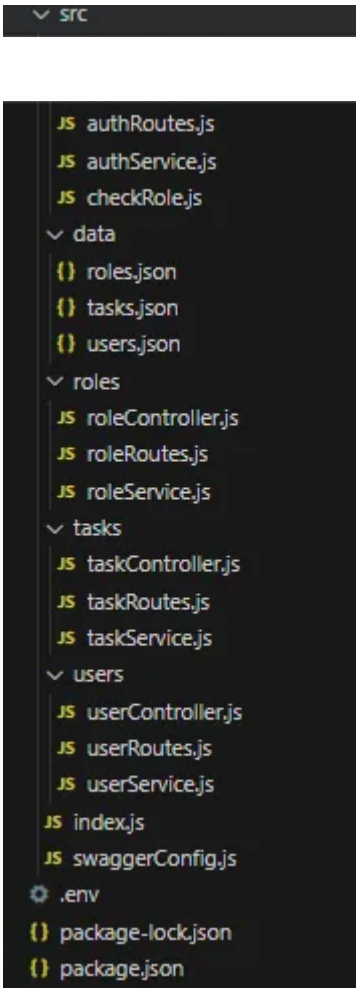
In this tutorial, we'll explore the development of a **secure RESTful API using Node.js**, organized with the **Model-View-Controller (MVC) architectural pattern.** This API will be structured into modular components, with separate routes and controllers for managing users, roles, and tasks, ensuring a clean and scalable codebase.

To enhance security, we'll integrate **JWT (JSON Web Token)** for authentication, allowing secure user login and access control based on roles. Each module — Users, Roles, Tasks, and Authentication — will be handled independently, making the API flexible and maintainable.

By the end of this tutorial, you'll have a robust understanding of how to build a secure, modular REST API that not only follows best practices in design but also provides essential features for real-world applications, including user authentication and role-based access control.

If you're new to REST API development or want to understand the foundational concepts, I recommend checking out the first part of this series: Creating a Basic REST API with Node.js, Swagger MVC. It covers the basics of setting up a REST API using Node.js and Swagger, providing the groundwork for the more advanced topics we'll explore in this tutorial.

Structure:



Code:

```
# Initialize a new Node.js project
npm init -y

# Install necessary packages
npm install express body-parser

# Install Swagger documentation tools
npm install swagger-ui-express swagger-jsdoc

# Install bcryptjs and jsonwebtoken for authentication
npm install bcryptjs jsonwebtoken

# Install nodemon as a development dependency
npm install --save-dev nodemon

# Install dotenv for environment variable management
npm install dotenv
```

package.json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "fs": "^0.0.1-security",
    "jsonwebtoken": "^9.0.2",
    "swagger-jsdoc": "^6.2.8",
    "swagger-ui-express": "^5.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

swaggerConfig.js:

```
// swaggerConfig.js

const swaggerJsdoc = require('swagger-jsdoc');
const swaggerUi = require('swagger-ui-express');

// Swagger definition
const swaggerDefinition = {
   openapi: '3.0.0',
   info: {
       title: 'My API',
       version: '1.0.0',
       description: 'This API provides CRUD operations for managing users, role
   },
   servers: [{
       url: 'http://localhost:3000', // Replace with your server URL
       description: 'Development server',
   }],
   components: {
```

```
        securitySchemes: {

                type: 'http',
                scheme: 'bearer',
                bearerFormat: 'JWT',
            },
        },
    },
    security: [{
        bearerAuth: []
    }],
};

// Options for the swagger-jsdoc
const options = {
    swaggerDefinition,
    // Paths to files containing OpenAPI definitions
    apis: ['./src/users/*.js','./src/roles/*.js','./src/auth/*.js','./src/tasks/
};

// Initialize swagger-jsdoc
const swaggerSpec = swaggerJsdoc(options);

module.exports = { swaggerUi, swaggerSpec };
```

index.js:

```
// index.js

const express = require('express');
const bodyParser = require('body-parser');
require('dotenv').config();  // Load environment variables

const taskRoutes = require('../src/tasks/taskRoutes');
const userRoutes = require('../src/users/userRoutes');
const roleRoutes = require('../src/roles/roleRoutes');
const authRoutes = require('../src/auth/authRoutes');
const { swaggerUi, swaggerSpec } = require('../src/swaggerConfig'); // Import S

const app = express();
const PORT = process.env.PORT || 3000;

app.use(bodyParser.json());

// Serve Swagger UI
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

// Auth Routes
```

```
app.use('/api/auth', authRoutes);

// Task Routes
app.use('/api/tasks', taskRoutes);

// User Routes
app.use('/api/users', userRoutes);

// Role Routes
app.use('/api/roles', roleRoutes);

app.listen(PORT, () => {
    console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

.env:

```
SECRET_KEY=your_secret_key_here
PORT=3000
```

roles.json:

```
[
    {
        "name": "admin",
        "status": true,
        "id": 1
    },
    {
        "name": "user",
        "status": true,
        "id": 2
    }
]
```

tasks.json:

```
[
    {
        "title": "Nueva Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
        "createdBy": 1,
        "assignedTo": 2,
        "id": 3
    },
    {
        "title": "nww Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
        "assignedTo": 3,
        "state": "pendiente",
        "id": 4
    },
    {
        "title": "nww Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
        "assignedTo": 3,
        "state": "pendiente",
        "id": 5
    },
    {
        "title": "nww Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
        "assignedTo": 3,
        "state": "pendiente",
        "id": 6
    },
    {
        "title": "nww Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
        "assignedTo": 3,
        "state": "pendiente",
        "id": 7
    },
    {
        "title": "nww Tarea",
        "description": "Descripción opcional de la tarea",
        "dueDate": "2024-08-08T05:00:00.000Z",
        "status": true,
```

```
        "assignedTo": 3,
        "id": 8
    }
]
```


users.json:


```
[
    {
        "username": "user4",
        "password": "$2a$08$y/T//JOL52OaeOxJbLs2Uu0uzzYdKZFY5xtayXXVJoalxkcvXn9
        "roleId": 1,
        "id": 1
    },
    {
        "username": "user5",
        "password": "$2a$08$GqT2Mg5UJAX.yp02N8eyN.Nw80VFnY7lvF7h5nXPZ7nZ.dCab.y
        "roleId": 2,
        "id": 3
    },
    {
        "username": "user6",
        "password": "$2a$08$tbu74IIlfTDtj6MzWAEwluxDmRFmfKCOGOEJLtu7qoGhA9iZKXF
        "roleId": 2,
        "id": 4
    },
    {
        "username": "user67",
        "password": "$2a$08$GtC9XZDWsqLjXmSohFNYhuKXe33/twp8H2VrgqYOJnqKmqaEtBE
        "roleId": 1,
        "id": 5
    },
    {
        "username": "usernew",
        "password": "$2a$08$dXAtEzF52jWBRnQLZXbhXOfW3aiFdedwM96.GGb/alJydlT8.S1
        "roleId": 1,
        "id": 6
    },
    {
        "username": "user637",
        "password": "$2a$08$PsHuc7Cct3c8asqCFTbLSuPyN5O.mvJIun25nOc69p5iGtTHFEr
        "roleId": 1,
        "id": 7
    }
]
```

authController.js:

```
//authController.js

const bcrypt = require('bcryptjs');

const { generateToken } = require('./authService');

const User = require('../users/userService');
const Role = require('../roles/roleService');

const register = (req, res) => {
    const { username, password, roleId } = req.body;
    if (!username || !password || !roleId) {
        return res.status(400).send({ message: 'Username, password, and role ID
    }
    const role = Role.getRoleById(roleId);
    if (!role) {
        return res.status(404).send({ message: 'Role not found.' });
    }
    const newUser = { username, password, roleId };
    const user = User.addUser(newUser);
    const token = generateToken(user);
    res.status(201).send({ auth: true, token });
};

const login = (req, res) => {
    const { username, password } = req.body;
    const user = User.getAllUsers().find(u => u.username === username);
    if (!user) {
        return res.status(404).send({ message: 'User not found.' });
    }
    const passwordIsValid = bcrypt.compareSync(password, user.password);
    if (!passwordIsValid) {
        return res.status(401).send({ auth: false, token: null });
    }
    const token = generateToken(user);
    res.status(200).send({ auth: true, token });
};

module.exports = { register, login };
```

authRoutes.js:

```
//authR

const express = require('express');
const router = express.Router();
const authController = require('./authController');

/**
 * @swagger
 * /api/auth/register:
 *   post:
 *     summary: Register a new user
 *     description: Creates a new user in the system.
 *     tags:
 *       - Authentication
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               username:
 *                 type: string
 *                 description: The user's username
 *               password:
 *                 type: string
 *                 description: The user's password
 *               roleId:
 *                 type: integer
 *                 description: The ID of the role assigned to the user
 *     responses:
 *       201:
 *         description: User created successfully
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 auth:
 *                   type: boolean
 *                 token:
 *                   type: string
 *       400:
 *         description: Missing username, password, or role ID
 *       404:
 *         description: Role not found
 *     security: []  # This route does not require bearer authentication
 */
router.post('/register', authController.register);

/**
```

```
 * @swagger
 * /ap
 *   post:
 *     summary: User login
 *     description: Authenticates a user and returns a JWT token.
 *     tags:
 *       - Authentication
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               username:
 *                 type: string
 *                 description: The user's username
 *               password:
 *                 type: string
 *                 description: The user's password
 *     responses:
 *       200:
 *         description: Successful login
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 auth:
 *                   type: boolean
 *                 token:
 *                   type: string
 *       401:
 *         description: Invalid password
 *       404:
 *         description: User not found
 *     security: []  # This route does not require bearer authentication
 */
router.post('/login', authController.login);

module.exports = router;
```

authService.js:

```
//authService.js

// Load environment variables
require('dotenv').config();
```

```
const
const SECRET_KEY = process.env.SECRET_KEY;

// Generates a JWT token with user details and a 1-hour expiration
function generateToken(user) {
    return jwt.sign({ id: user.id, username: user.username, roleId: user.roleId
}

// Middleware to verify the JWT token from the Authorization header
function verifyToken(req, res, next) {
    const token = req.headers['authorization'];
    if (!token) {
        return res.status(403).send({ message: 'No token provided.' });
    }
    const tokenParts = token.split(' ');
    if (tokenParts.length !== 2 || tokenParts[0] !== 'Bearer') {
        return res.status(403).send({ message: 'Invalid token format.' });
    }
    jwt.verify(tokenParts[1], SECRET_KEY, (err, decoded) => {
        if (err) {
            return res.status(500).send({ message: 'Failed to authenticate token
        }
        req.user = { id: decoded.id, username: decoded.username, roleId: decode
        next();
    });
}

module.exports = { generateToken, verifyToken };
```

checkRole.js:

```
//checkRole.js

//Middleware to restrict access based on roleId

function checkRole(roleId) {
    return (req, res, next) => {
        if (req.user.roleId !== roleId) {
            return res.status(403).send({ message: 'Access denied. Insufficient
        }
        next();
    };
}
```

```
module.
```

roleController.js:

```
//roleController.js

const Role = require('./roleService');

module.exports = {

    getAllRoles(req, res) {
        try {
            const roles = Role.getAllRoles();
            res.status(200).json(roles);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving roles', error: er
        }
    },

    getRoleById(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid role ID' });
            }
            const role = Role.getRoleById(id);
            if (!role) {
                return res.status(404).json({ message: 'Role not found' });
            }
            res.status(200).json(role);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving role', error: err
        }
    },

    addRole(req, res) {
        try {
            const { name, status } = req.body;
            if (!name || typeof status !== 'boolean') {
                return res.status(400).json({ message: 'Invalid role data' });
            }
            //Check if the role name already exists
            const existingRole = Role.getAllRoles().find(role => role.name ===
            if (existingRole) {
                return res.status(400).json({ message: 'Role name already exist
            }
```

```
            const newRole = { name, status };

            res.status(201).json(role);
        } catch (error) {
            res.status(500).json({ message: 'Error adding role', error: error.m
        }
    },

    updateRole(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid role ID' });
            }
            const { name, status } = req.body;
            if (!name || typeof status !== 'boolean') {
                return res.status(400).json({ message: 'Invalid role data' });
            }

            //Check if the role name already exists and is not the name of the
            const existingRole = Role.getAllRoles().find(role => role.name ===
            if (existingRole) {
                return res.status(400).json({ message: 'Role name already exist
            }

            const updatedRole = { name, status };
            const role = Role.updateRole(id, updatedRole);
            if (!role) {
                return res.status(404).json({ message: 'Role not found' });
            }
            res.status(200).json(role);
        } catch (error) {
            res.status(500).json({ message: 'Error updating role', error: error
        }
    },

    deleteRole(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid role ID' });
            }
            const role = Role.deleteRole(id);
            if (!role) {
                return res.status(404).json({ message: 'Role not found' });
            }
            res.status(200).json({ message: 'Role deleted successfully' });
        } catch (error) {
            res.status(500).json({ message: 'Error deleting role', error: error
        }
    }
};
```

roleRoutes.js:

```
//roleRoutes.js

const express = require('express');
const router = express.Router();
const roleController = require('./roleController');
const { verifyToken } = require('../auth/authService');

/**
 * @swagger
 * tags:
 *   name: Roles
 *   description: Role management and administration
 */

/**
 * @swagger
 * components:
 *   schemas:
 *     Role:
 *       type: object
 *       properties:
 *         id:
 *           type: integer
 *           description: Role ID
 *         name:
 *           type: string
 *           description: Name of the role
 *         status:
 *           type: boolean
 *           description: Status of the role (active/inactive)
 *       required:
 *         - name
 *         - status
 */

/**
 * @swagger
 * /api/roles:
 *   get:
 *     summary: Get all roles
 *     tags:
 *       - Roles
 *     responses:
 *       200:
```

```
 *         description: Returns all roles
 *
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/Role'
 */
router.get('/', verifyToken, roleController.getAllRoles);

/**
 * @swagger
 * /api/roles/{id}:
 *   get:
 *     summary: Get a role by ID
 *     tags:
 *       - Roles
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the role to get
 *         example: 1
 *     responses:
 *       200:
 *         description: Returns the specified role
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Role'
 *       404:
 *         description: Role not found
 */
router.get('/:id', verifyToken, roleController.getRoleById);

/**
 * @swagger
 * /api/roles:
 *   post:
 *     summary: Add a new role
 *     tags:
 *       - Roles
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Role'
 *           examples:
 *             roleExample:
 *               summary: Example of a new role
```

```
 *               value:
 *
 *                 status: true
 *     responses:
 *       201:
 *         description: New role created
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Role'
 */
router.post('/', roleController.addRole);

/**
 * @swagger
 * /api/roles/{id}:
 *   patch:
 *     summary: Update an existing role
 *     tags:
 *       - Roles
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the role to update
 *         example: 1
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Role'
 *           examples:
 *             updateRoleExample:
 *               summary: Example of an updated role
 *               value:
 *                 name: "updatedrole"
 *                 status: false
 *     responses:
 *       200:
 *         description: Role updated successfully
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Role'
 *       404:
 *         description: Role not found
 */
router.patch('/:id', verifyToken, roleController.updateRole);

/**
```

```
 * @swagger
 * /ap
 *   delete:
 *     summary: Delete a role
 *     tags:
 *       - Roles
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the role to delete
 *         example: 1
 *     responses:
 *       200:
 *         description: Role deleted successfully
 *       404:
 *         description: Role not found
 */
router.delete('/:id', verifyToken, roleController.deleteRole);

module.exports = router;
```

roleService.js:

```
//roleService.js

const fs = require('fs');

// Path to the JSON file where roles will be stored
const filePath = './src/data/roles.json';

// Function to load roles from the JSON file
function loadRoles() {
    try {
        const data = fs.readFileSync(filePath);
        return JSON.parse(data);
    } catch (error) {
        // If there's an error reading the file, return an empty array
        return [];
    }
}

// Function to save roles to the JSON file
function saveRoles(roles) {
    fs.writeFileSync(filePath, JSON.stringify(roles, null, 4));
}
```

```
module

    // Retrieve all roles
    getAllRoles() {
        return loadRoles();
    },

    // Retrieve a role by its ID
    getRoleById(id) {
        const roles = loadRoles();
        return roles.find(role => role.id === id);
    },

    // Add a new role
    addRole(role) {
        const roles = loadRoles();
        role.id = roles.length > 0 ? roles[roles.length - 1].id + 1 : 1;
        roles.push(role);
        saveRoles(roles);
        return role;
    },

    // Update an existing role
    updateRole(id, updatedRole) {
        const roles = loadRoles();
        const roleIndex = roles.findIndex(role => role.id === id);
        if (roleIndex === -1) {
            return null;
        }
        roles[roleIndex] = { ...roles[roleIndex], ...updatedRole };
        saveRoles(roles);
        return roles[roleIndex];
    },

    // Delete a role by its ID
    deleteRole(id) {
        const roles = loadRoles();
        const roleIndex = roles.findIndex(role => role.id === id);
        if (roleIndex === -1) {
            return null;
        }
        const deletedRole = roles.splice(roleIndex, 1);
        saveRoles(roles);
        return deletedRole;
    }

};
```

taskController.js:

```
//task(

const Task = require('./taskService');
const User = require('../users/userService');

module.exports = {
    getAllTasks(req, res) {
        try {
            const tasks = Task.getAllTasks();
            tasks.forEach(task => {
                if (task.dueDate) {
                    const date = new Date(task.dueDate);
                    task.dueDate = `${date.getDate().toString().padStart(2, '0')
                }
            });
            res.status(200).json(tasks);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving tasks', error: err
        }
    },

    getTaskById(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid task ID' });
            }
            const task = Task.getTaskById(id);
            if (!task) {
                return res.status(404).json({ message: 'Task not found' });
            }
            if (task.dueDate) {
                const date = new Date(task.dueDate);
                task.dueDate = `${date.getDate().toString().padStart(2, '0')}/${
            }
            res.status(200).json(task);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving task', error: erro
        }
    },

    addTask(req, res) {
        try {
            const { title, description, dueDate, status, assignedTo, state } = r
            if (!title || typeof status !== 'boolean' || !['pendiente', 'en proc
                return res.status(400).json({ message: 'Invalid task data' });
            }


            if (assignedTo && !User.getUserById(assignedTo)) {
                return res.status(404).json({ message: 'Assigned user not found'
```

```
        }


        const [day, month, year] = dueDate.split('/').map(Number);
        const formattedDueDate = new Date(year, month - 1, day);
        if (isNaN(formattedDueDate)) {
            return res.status(400).json({ message: 'Invalid due date' });
        }

        const newTask = {
            title,
            description,
            dueDate: formattedDueDate,
            status,
            createdBy: req.userId,
            assignedTo,
            state
        };

        const task = Task.addTask(newTask);
        task.dueDate = dueDate;
        res.status(201).json(task);
    } catch (error) {
        res.status(500).json({ message: 'Error adding task', error: error.me
    }
},

updateTask(req, res) {
    try {
        const id = parseInt(req.params.id);
        if (isNaN(id)) {
            return res.status(400).json({ message: 'Invalid task ID' });
        }
        const { title, description, dueDate, status, assignedTo, state } = r
        if (!title || typeof status !== 'boolean' || !['pendiente', 'en proc
            return res.status(400).json({ message: 'Invalid task data' });
        }


        if (assignedTo && !User.getUserById(assignedTo)) {
            return res.status(404).json({ message: 'Assigned user not found'
        }


        const [day, month, year] = dueDate.split('/').map(Number);
        const formattedDueDate = new Date(year, month - 1, day);
        if (isNaN(formattedDueDate)) {
            return res.status(400).json({ message: 'Invalid due date' });
        }

        const updatedTask = {
            title,
            description,
```

```
                dueDate: formattedDueDate,

                assignedTo,
                state
            };
            const task = Task.updateTask(id, updatedTask);
            if (!task) {
                return res.status(404).json({ message: 'Task not found' });
            }
            task.dueDate = dueDate;
            res.status(200).json(task);
        } catch (error) {
            res.status(500).json({ message: 'Error updating task', error: error.
        }
    },

    deleteTask(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid task ID' });
            }
            const task = Task.deleteTask(id);
            if (!task) {
                return res.status(404).json({ message: 'Task not found' });
            }
            res.status(200).json({ message: 'Task deleted successfully' });
        } catch (error) {
            res.status(500).json({ message: 'Error deleting task', error: error.
        }
    }
};
```

taskRoutes.js:

```
//taskRoutes.js

const express = require('express');
const router = express.Router();
const taskController = require('./taskController');
const { verifyToken } = require('../auth/authService');

/**
 * @swagger
 * tags:
 *   name: Tasks
 *   description: Task management and administration
 */
```

```
/**
 * @swagger
 * components:
 *   schemas:
 *     Task:
 *       type: object
 *       properties:
 *         id:
 *           type: integer
 *           description: Task ID
 *         title:
 *           type: string
 *           description: Title of the task
 *         description:
 *           type: string
 *           description: Description of the task (optional)
 *         dueDate:
 *           type: string
 *           format: date
 *           description: Due date of the task
 *         status:
 *           type: boolean
 *           description: Status of the task (completed/pending)
 *         assignedTo:
 *           type: integer
 *           description: ID of the user the task is assigned to
 *         state:
 *           type: string
 *           description: Current state of the task (e.g., 'pendiente', 'en prc
 *       required:
 *         - title
 *         - dueDate
 *         - status
 *         - assignedTo
 *         - state
 */

/**
 * @swagger
 * /api/tasks:
 *   get:
 *     summary: Get all tasks
 *     tags:
 *       - Tasks
 *     responses:
 *       200:
 *         description: Returns all tasks
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
```

```
 *                 $ref: '#/components/schemas/Task'
 */
router.get('/', verifyToken, taskController.getAllTasks);

/**
 * @swagger
 * /api/tasks/{id}:
 *   get:
 *     summary: Get a task by ID
 *     tags:
 *       - Tasks
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the task to get
 *         example: 1
 *     responses:
 *       200:
 *         description: Returns the specified task
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Task'
 *       404:
 *         description: Task not found
 */
router.get('/:id', verifyToken, taskController.getTaskById);

/**
 * @swagger
 * /api/tasks:
 *   post:
 *     summary: Add a new task
 *     tags:
 *       - Tasks
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Task'
 *           examples:
 *             taskExample:
 *               summary: Example of a new task
 *               value:
 *                 title: "nww Tarea"
 *                 description: "Descripción opcional de la tarea"
 *                 dueDate: "08/08/2024"
 *                 status: true
 *                 assignedTo: 2
```

```
 *                 state: "pendiente"
 *
 *       201:
 *         description: New task created
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Task'
 */
router.post('/', verifyToken, taskController.addTask);

/**
 * @swagger
 * /api/tasks/{id}:
 *   patch:
 *     summary: Update an existing task
 *     tags:
 *       - Tasks
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the task to update
 *         example: 1
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/Task'
 *           examples:
 *             updateTaskExample:
 *               summary: Example of an updated task
 *               value:
 *                 title: "Tarea actualizada"
 *                 description: "Nueva descripción opcional"
 *                 dueDate: "09/09/2024"
 *                 status: false
 *                 assignedTo: 3
 *                 state: "en progreso"
 *     responses:
 *       200:
 *         description: Task updated successfully
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/Task'
 *       404:
 *         description: Task not found
 */
router.patch('/:id', verifyToken, taskController.updateTask);
```

```
/**
 * @swagger
 * /api/tasks/{id}:
 *   delete:
 *     summary: Delete a task
 *     tags:
 *       - Tasks
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the task to delete
 *         example: 1
 *     responses:
 *       200:
 *         description: Task deleted successfully
 *       404:
 *         description: Task not found
 */
router.delete('/:id', verifyToken, taskController.deleteTask);

module.exports = router;
```

taskService.js:

```
//taskService.js

const fs = require('fs');

// Path to the JSON file where tasks will be stored
const filePath = './src/data/tasks.json';

// Function to load tasks from the JSON file
function loadTasks() {
    try {
        const data = fs.readFileSync(filePath);
        return JSON.parse(data);
    } catch (error) {
        // If there's an error reading the file, return an empty array
        return [];
    }
}

// Function to save tasks to the JSON file
function saveTasks(tasks) {
```

```
    fs.writeFileSync(filePath, JSON.stringify(tasks, null, 4));
}

// Function to format the date as 'dd/MM/yyyy'
function formatDate(date) {
    const day = date.getDate().toString().padStart(2, '0');
    const month = (date.getMonth() + 1).toString().padStart(2, '0');
    const year = date.getFullYear();
    return `${day}/${month}/${year}`;
}

module.exports = {

    // Retrieve all tasks
    getAllTasks() {
        return loadTasks();
    },

    // Retrieve a task by its ID
    getTaskById(id) {
        const tasks = loadTasks();
        return tasks.find(task => task.id === id);
    },

    // Add a new task
    addTask(task) {
        const tasks = loadTasks();
        task.id = tasks.length > 0 ? tasks[tasks.length - 1].id + 1 : 1;
        tasks.push(task);
        saveTasks(tasks);
        return task;
    },

    // Update an existing task
    updateTask(id, updatedTask) {
        const tasks = loadTasks();
        const taskIndex = tasks.findIndex(task => task.id === id);
        if (taskIndex === -1) {
            return null;
        }
        tasks[taskIndex] = { ...tasks[taskIndex], ...updatedTask };
        saveTasks(tasks);
        return tasks[taskIndex];
    },

    // Delete a task by its ID
    deleteTask(id) {
        const tasks = loadTasks();
        const taskIndex = tasks.findIndex(task => task.id === id);
        if (taskIndex === -1) {
            return null;
        }
        const deletedTask = tasks.splice(taskIndex, 1);
```

```
        saveTasks(tasks);

    }

};
```

userController.js:

```
//userController.js

const User = require('./userService');
const Role = require('../roles/roleService');

module.exports = {

    getAllUsers(req, res) {
        try {
            const users = User.getAllUsers();
            res.status(200).json(users);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving users', error: er
        }
    },

    getUserById(req, res) {
        try {
            const id = parseInt(req.params.id);
            if (isNaN(id)) {
                return res.status(400).json({ message: 'Invalid user ID' });
            }
            const user = User.getUserById(id);
            if (!user) {
                return res.status(404).json({ message: 'User not found' });
            }
            res.status(200).json(user);
        } catch (error) {
            res.status(500).json({ message: 'Error retrieving user', error: err
        }
    },

    addUser(req, res) {
        try {
            const { username, password, roleId } = req.body;
            if (!username || !password || !roleId) {
                return res.status(400).json({ message: 'Invalid user data' });
            }
            //Check if the username already exists
            const existingUser = User.getAllUsers().find(user => user.username
```

```
        if (existingUser) {
                                                                    eady exists
        }
        // Check if the roleId is valid
        const role = Role.getRoleById(roleId);
        if (!role) {
            return res.status(404).json({ message: 'Role not found' });
        }
        const newUser = { username, password, roleId };
        const user = User.addUser(newUser);
        res.status(201).json(user);
    } catch (error) {
        res.status(500).json({ message: 'Error adding user', error: error.m
    }
},

updateUser(req, res) {
    try {
        const id = parseInt(req.params.id);
        if (isNaN(id)) {
            return res.status(400).json({ message: 'Invalid user ID' });
        }
        const { username, password, roleId } = req.body;
        if (!username || !password || !roleId) {
            return res.status(400).json({ message: 'Invalid user data' });
        }
        //Check if the username already exists and is not the one of the us
        const existingUser = User.getAllUsers().find(user => user.username
        if (existingUser) {
            return res.status(400).json({ message: 'Username already exists
        }
        // Check if the roleId is valid
        const role = Role.getRoleById(roleId);
        if (!role) {
            return res.status(404).json({ message: 'Role not found' });
        }
        const updatedUser = { username, password, roleId };
        const user = User.updateUser(id, updatedUser);
        if (!user) {
            return res.status(404).json({ message: 'User not found' });
        }
        res.status(200).json(user);
    } catch (error) {
        res.status(500).json({ message: 'Error updating user', error: error
    }
},

deleteUser(req, res) {
    try {
        const id = parseInt(req.params.id);
        if (isNaN(id)) {
            return res.status(400).json({ message: 'Invalid user ID' });
        }
```

```
            const user = User.deleteUser(id);

                return res.status(404).json({ message: 'User not found' });
            }
            res.status(200).json({ message: 'User deleted successfully' });
        } catch (error) {
            res.status(500).json({ message: 'Error deleting user', error: error
        }
    }
};
```

userRoutes.js:

```
//userRoutes.js

const express = require('express');
const router = express.Router();
const userController = require('./userController');
const { verifyToken } = require('../auth/authService');
const { checkRole } = require('../auth/checkRole');

/**
 * @swagger
 * tags:
 *   name: Users
 *   description: User management and administration
 */

/**
 * @swagger
 * components:
 *   schemas:
 *     User:
 *       type: object
 *       properties:
 *         id:
 *           type: integer
 *           description: User ID
 *         username:
 *           type: string
 *           description: Username of the user
 *         password:
 *           type: string
 *           description: Password of the user
 *         roleId:
 *           type: integer
 *           description: ID of the role assigned to the user
 *       required:
```

```
 *         - username
 *
 *         - roleId
 */

/**
 * @swagger
 * /api/users:
 *   get:
 *     summary: Get all users
 *     tags:
 *       - Users
 *     responses:
 *       200:
 *         description: Returns all users
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/User'
 */
router.get('/', verifyToken, checkRole(1), userController.getAllUsers);

/**
 * @swagger
 * /api/users/{id}:
 *   get:
 *     summary: Get a user by ID
 *     tags:
 *       - Users
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the user to get
 *     responses:
 *       200:
 *         description: Returns the specified user
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/User'
 *       404:
 *         description: User not found
 */
router.get('/:id', verifyToken, checkRole(1), userController.getUserById);

/**
 * @swagger
 * /api/users:
```

```
 *   post:
 *
 *     tags:
 *       - Users
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/User'
 *     responses:
 *       201:
 *         description: New user created
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/User'
 */
router.post('/', userController.addUser);

/**
 * @swagger
 * /api/users/{id}:
 *   patch:
 *     summary: Update an existing user
 *     tags:
 *       - Users
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the user to update
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             $ref: '#/components/schemas/User'
 *     responses:
 *       200:
 *         description: User updated successfully
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/User'
 *       404:
 *         description: User not found
 */
router.patch('/:id', verifyToken, userController.updateUser);

/**
```

```
 * @swagger
 * /ap
 *   delete:
 *     summary: Delete a user
 *     tags:
 *       - Users
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: integer
 *         required: true
 *         description: Numeric ID of the user to delete
 *     responses:
 *       200:
 *         description: User deleted successfully
 *       404:
 *         description: User not found
 */
router.delete('/:id', verifyToken, userController.deleteUser);

module.exports = router;
```

userService.js:

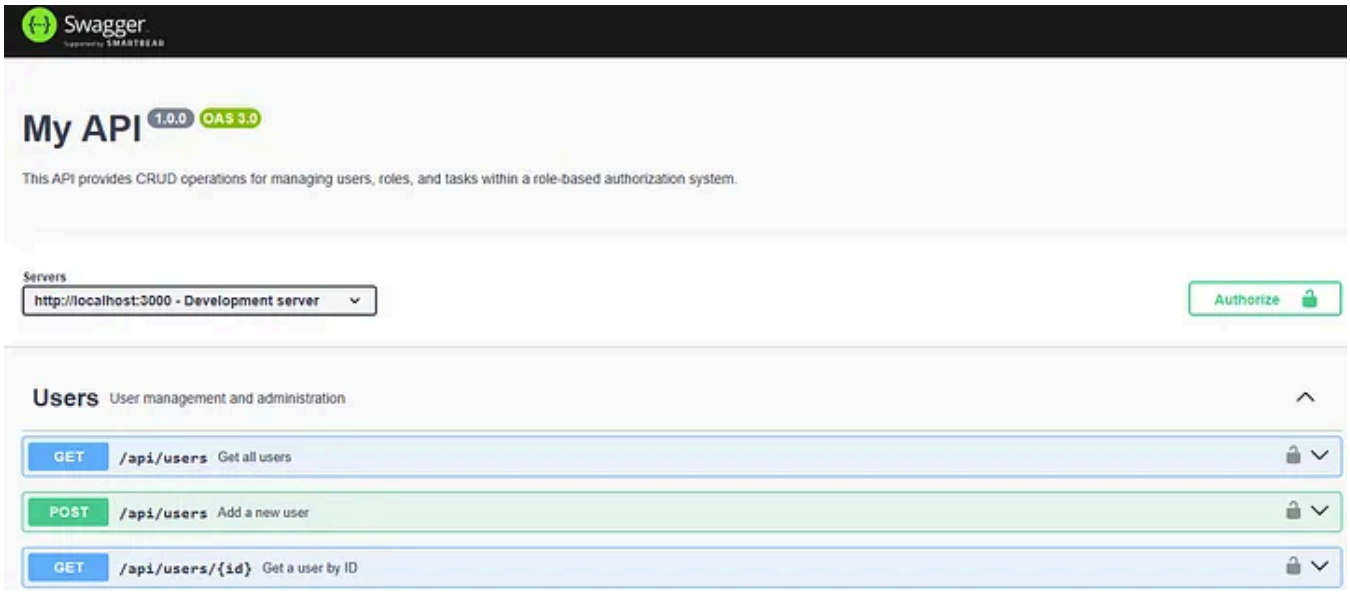
```
//userService.js

const fs = require('fs');
const bcrypt = require('bcryptjs');

// Path to the JSON file where users will be stored
const filePath = './src/data/users.json';

// Function to load users from the JSON file
function loadUsers() {
    try {
        const data = fs.readFileSync(filePath);
        return JSON.parse(data);
    } catch (error) {
        // If there's an error reading the file, return an empty array
        return [];
    }
}

// Function to save users to the JSON file
function saveUsers(users) {
    fs.writeFileSync(filePath, JSON.stringify(users, null, 4));
}
```

```
module

    // Retrieve all users
    getAllUsers() {
        return loadUsers();
    },

    // Retrieve a user by their ID
    getUserById(id) {
        const users = loadUsers();
        return users.find(user => user.id === id);
    },

    // Add a new user
    addUser(user) {
        const users = loadUsers();
        user.id = users.length > 0 ? users[users.length - 1].id + 1 : 1;
        user.password = bcrypt.hashSync(user.password, 8); // Encrypt the passw
        users.push(user);
        saveUsers(users);
        return user;
    },

    // Update an existing user
    updateUser(id, updatedUser) {
        const users = loadUsers();
        const userIndex = users.findIndex(user => user.id === id);
        if (userIndex === -1) {
            return null;
        }
        updatedUser.password = bcrypt.hashSync(updatedUser.password, 8); // Enc
        users[userIndex] = { ...users[userIndex], ...updatedUser };
        saveUsers(users);
        return users[userIndex];
    },

    // Delete a user by their ID
    deleteUser(id) {
        const users = loadUsers();
        const userIndex = users.findIndex(user => user.id === id);
        if (userIndex === -1) {
            return null;
        }
        const deletedUser = users.splice(userIndex, 1);
        saveUsers(users);
        return deletedUser;
    }

};
```

```
npm run
```

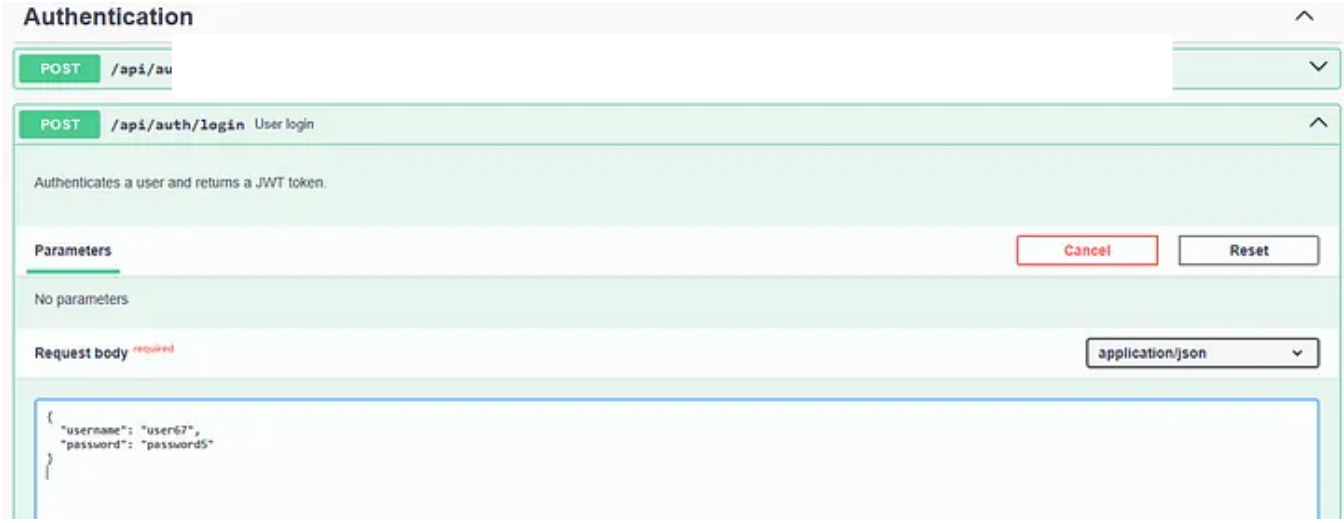
```
> backend@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
Servidor corriendo en http://localhost:3000
```

http://localhost:3000/api-docs/

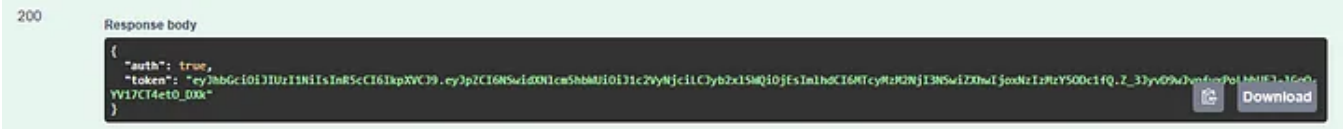


http://localhost:3000/api/users/

To access the routes, authentication is required.



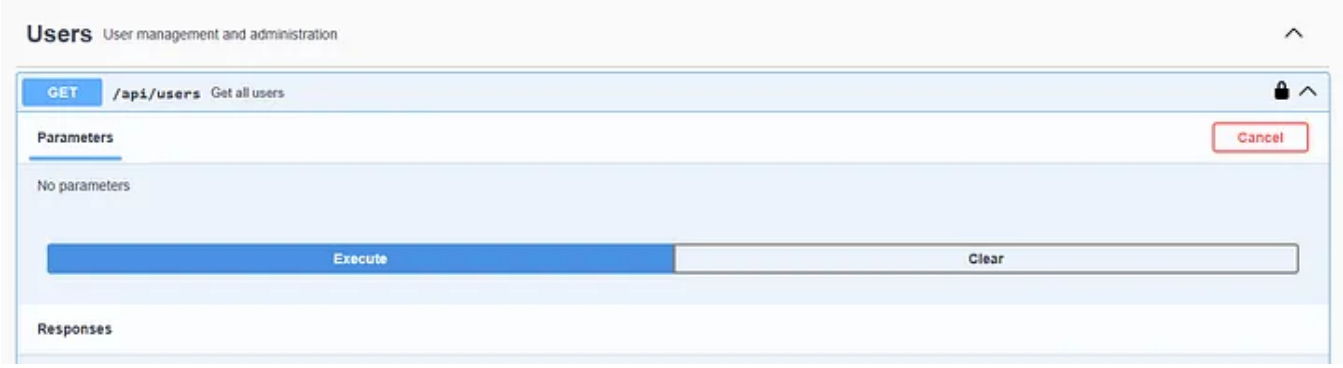
In the authentication section of the API, we send the access credentials, and if they are valid, the server will return an access token that can be used to authenticate future requests.

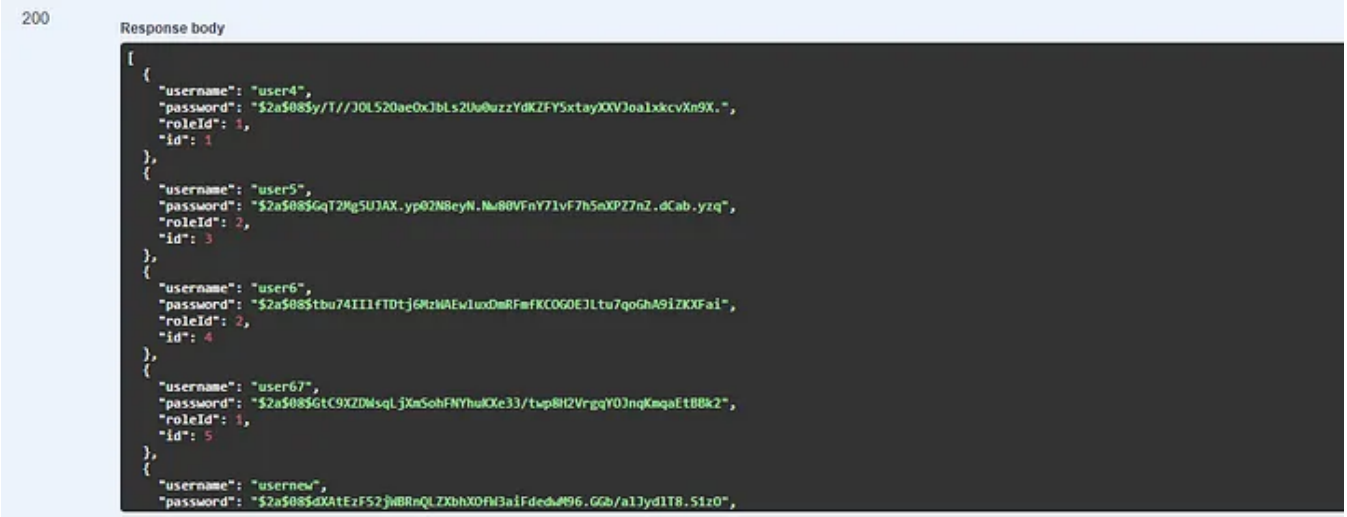


Copy the access token, press the authorization button.

Paste the token, and click “Authorize” and “Close”. With this, the authentication is complete and we should be allowed to see the information.

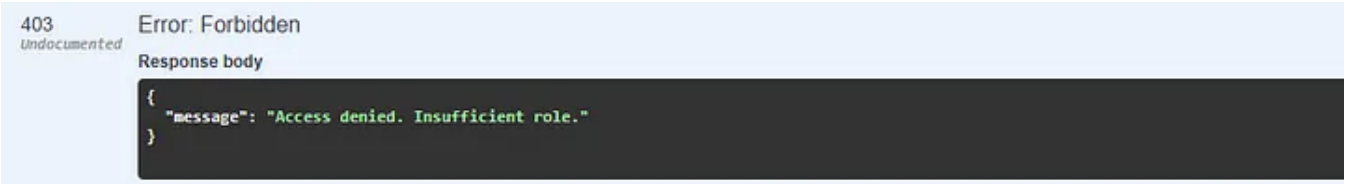
We test the users path to verify access





Displays the information, which means that verification and authentication are working correctly.

However, if we try to access a route without the required role, an error will appear indicating that the role does not have permission to access that route.



In conclusion, this is a project consisting of three tables: users, roles and tasks, stored in memory and managed through JSON files locally. The project implements Role-Based Access Control (RBAC), ensuring that only users with the appropriate permissions can access and perform specific actions on the various API paths. This approach provides secure and well-defined access control.

To view the code in the repository, go to the following link:

https://github.com/kadhir03/RoleBasedAccess-JSONAPI

Don’t hesitate to give more applause 👏🏻 and share the article with whoever you want. As always, I appreciate your support, I will continue with more tutorials.

Jwt | Role Based Access Control | Expressjs | Nodejs | Mvc

Follow

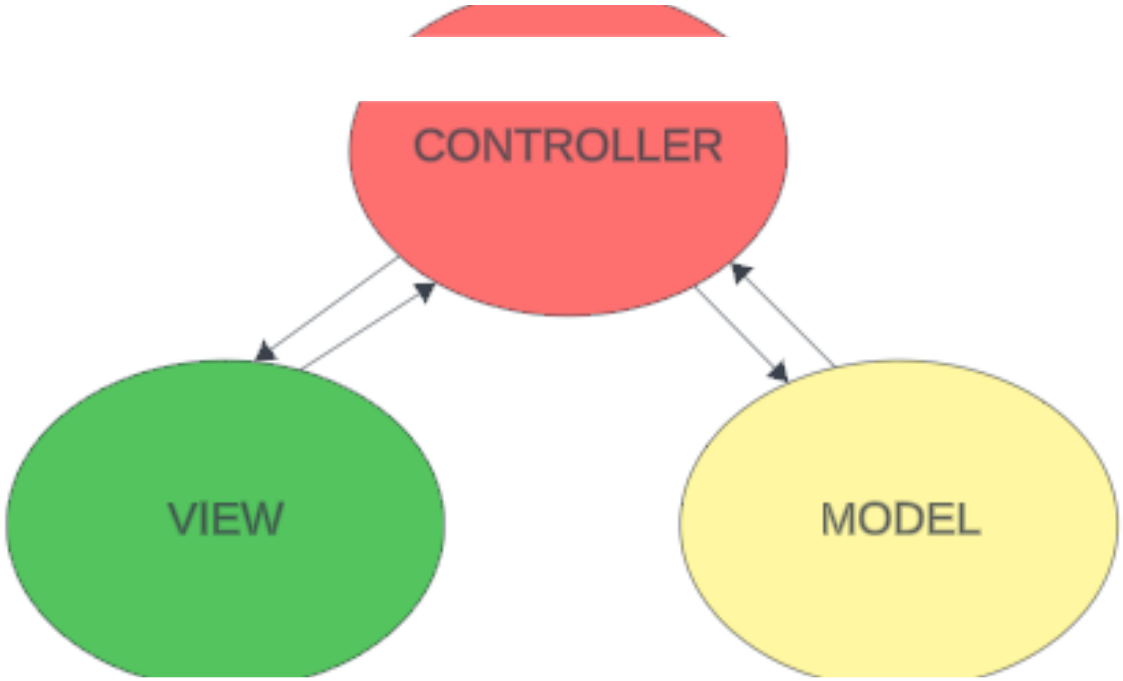
Written by Alberto R.

5 Followers

Final-semester Systems Engineering student, passionate about web/software development, exploring new technologies, creative solutions, and organizing activities

---

More from Alberto R.



Alberto R. in Dev Genius

Creating a basic REST API with TypeScript, Node.js, Swagger MVC

In this tutorial, we will explore developing a RESTful API using the Model-View-Controller (MVC) architectural pattern with TypeScript. MVC...

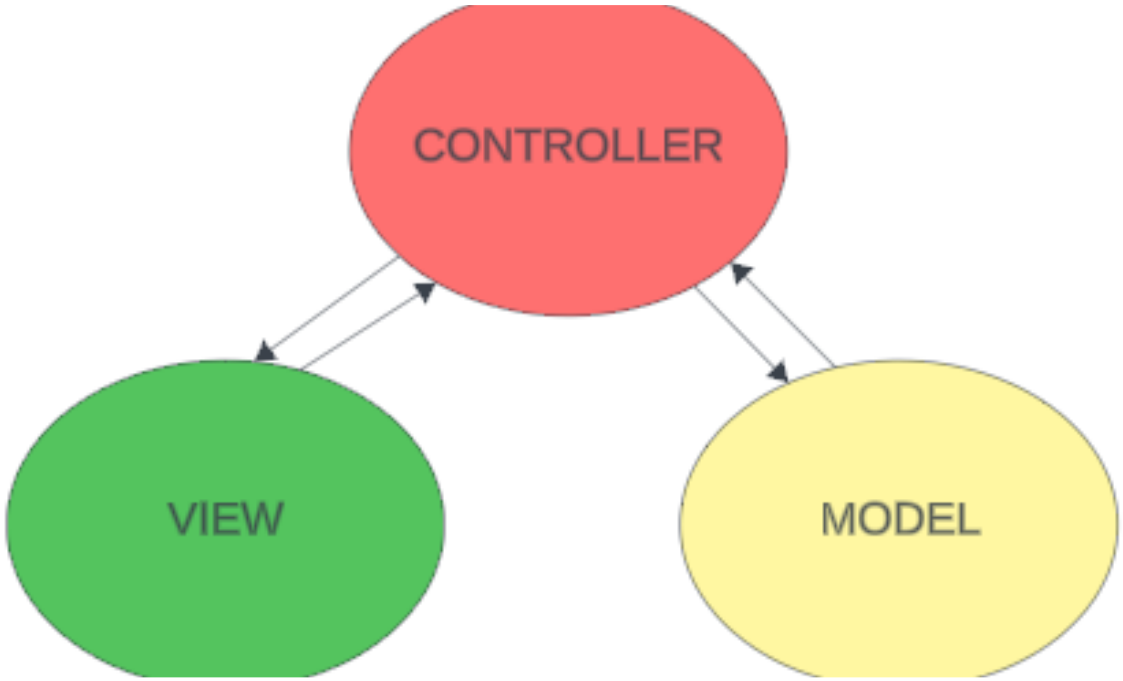
May 13

Alberto R. in Dev Genius

Create a Basic REST API with Native PHP, Docker, PostgreSQL, and pgAdmin 4

This tutorial aims to create a basic REST API using native PHP, Docker, PostgreSQL and PgAdmin4 a

May 29 1

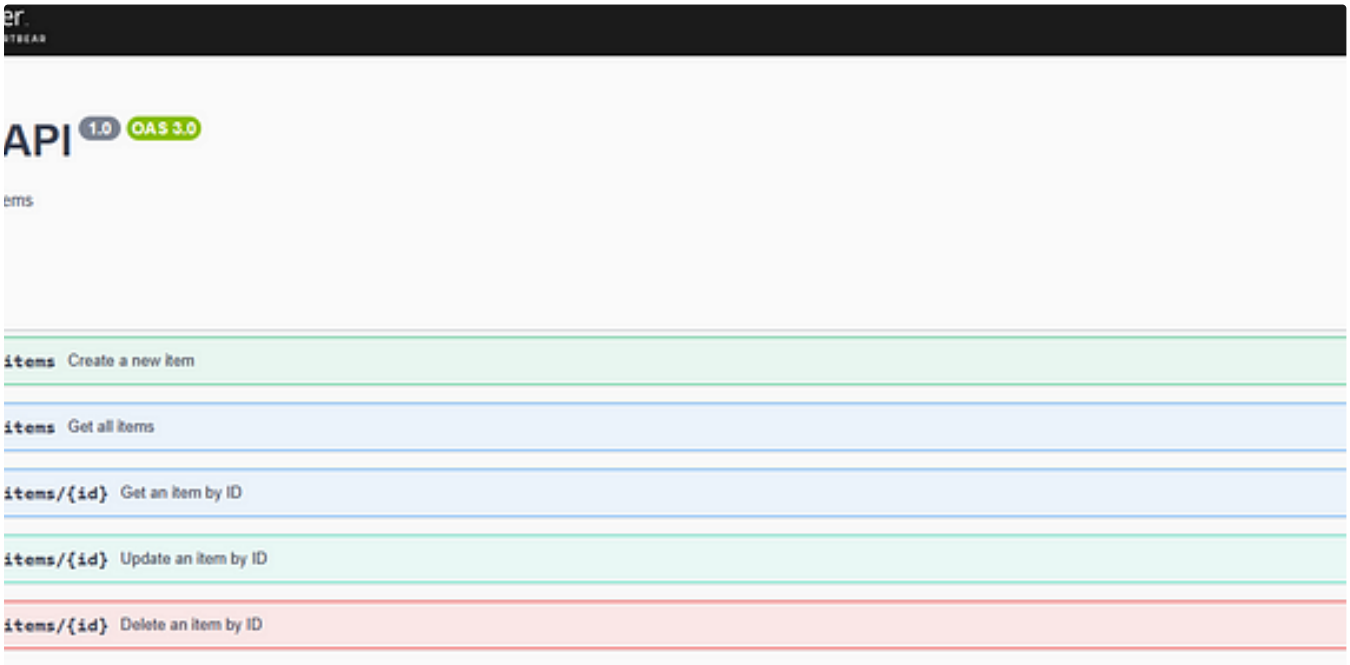


Alberto R. in Dev Genius

Creating a basic REST API with Node.js, Swagger MVC

In this tutorial, we'll explore the development of a RESTful API using the Model-View-Controller (MVC) architectural pattern. MVC divides...

May 13 1



Alberto R. in Dev Genius

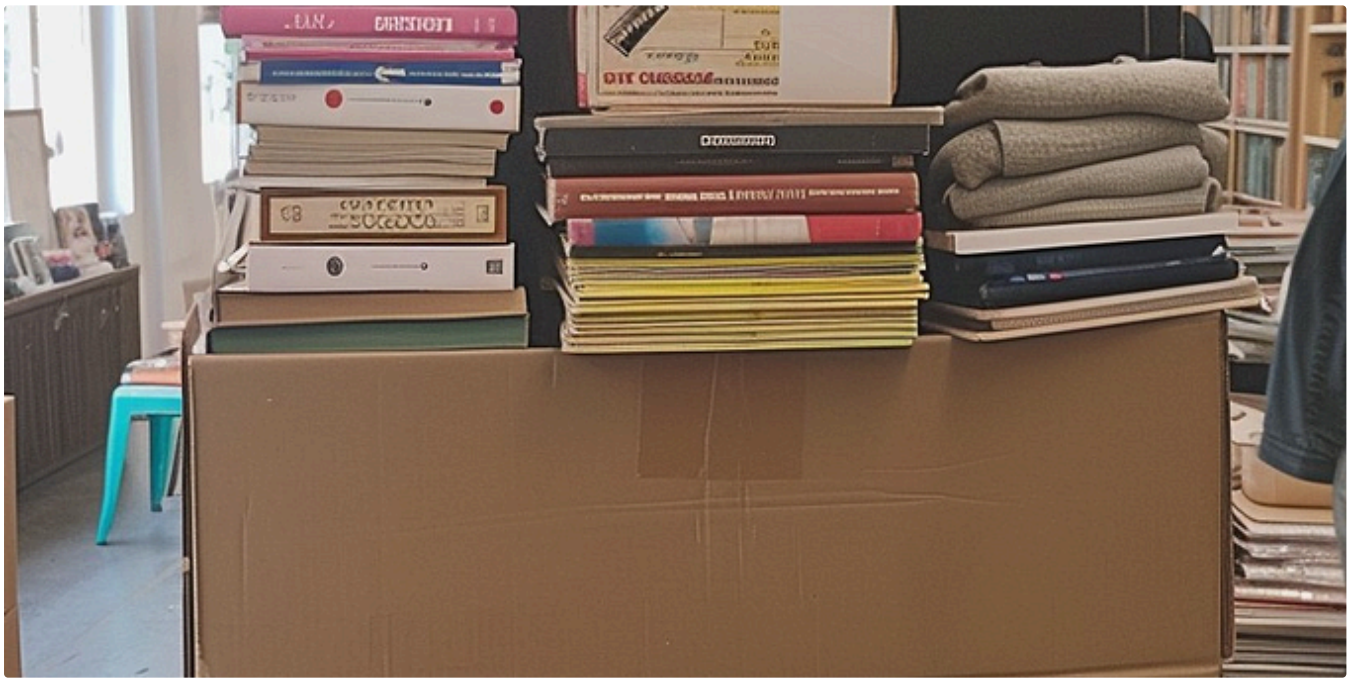
Creating a Basic Modular REST API with NestJS and Swagger

In this tutorial, we'll explore the development of a RESTful API using the NestJS framework with a modular architecture. NestJS leverages...

May 28

See all from Alberto R.

Recommended from Medium



Ankita Kolhe in Must-Read Articles in a Minute

Load Balancer in minute?

Load balancer is a device or software that distributes incoming network traffic across multiple servers to ensure no single server is...

Aug 5 4

Oliver Foster in Stackademic

Optimizing MySQL Queries from 190 Seconds to 1 Second for Tens of Millions of Records

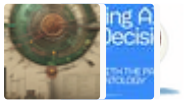
My article is open to everyone; non-member readers can click this link to read the full text.

Jul 24 585 7

Lists

Stories to Help You Grow as a Software Developer

19 stories · 1266 saves



data science and AI

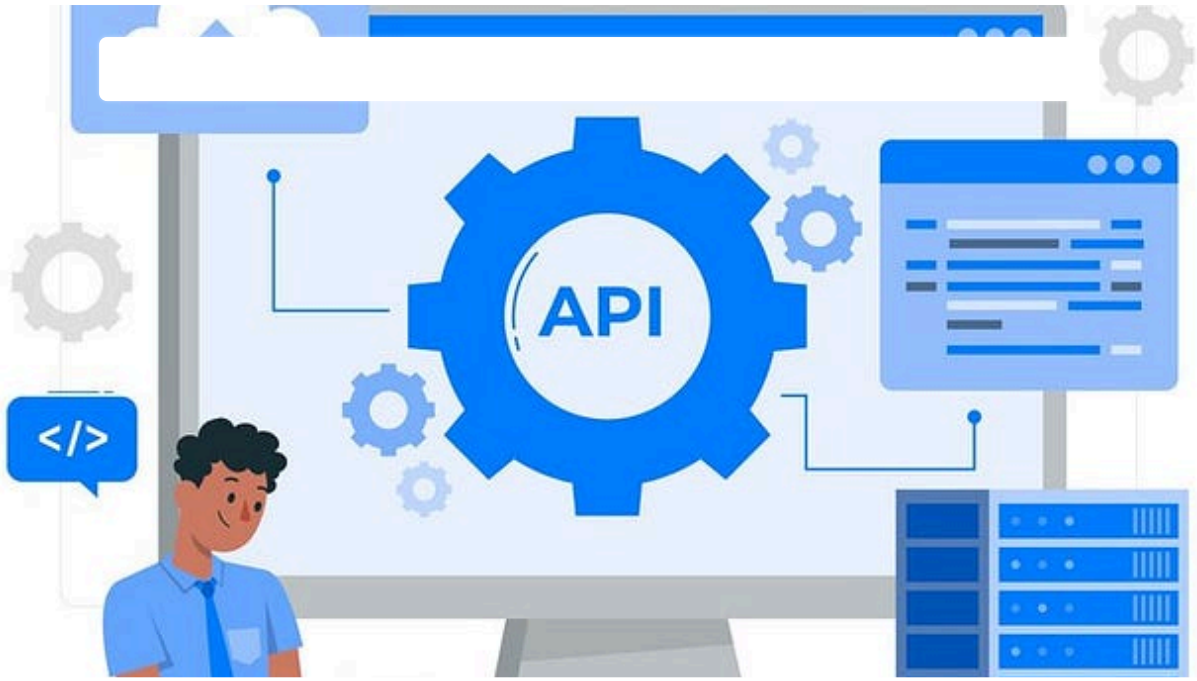
40 stories · 212 saves

Louis Trinh

Advanced Node.js API Logging with Winston and Morgan

Installation:

Apr 5 28



Fidelis Adewoye

Building Scalable APIs with Node.js, Express, and TypeScript: A Clean Architecture Approach

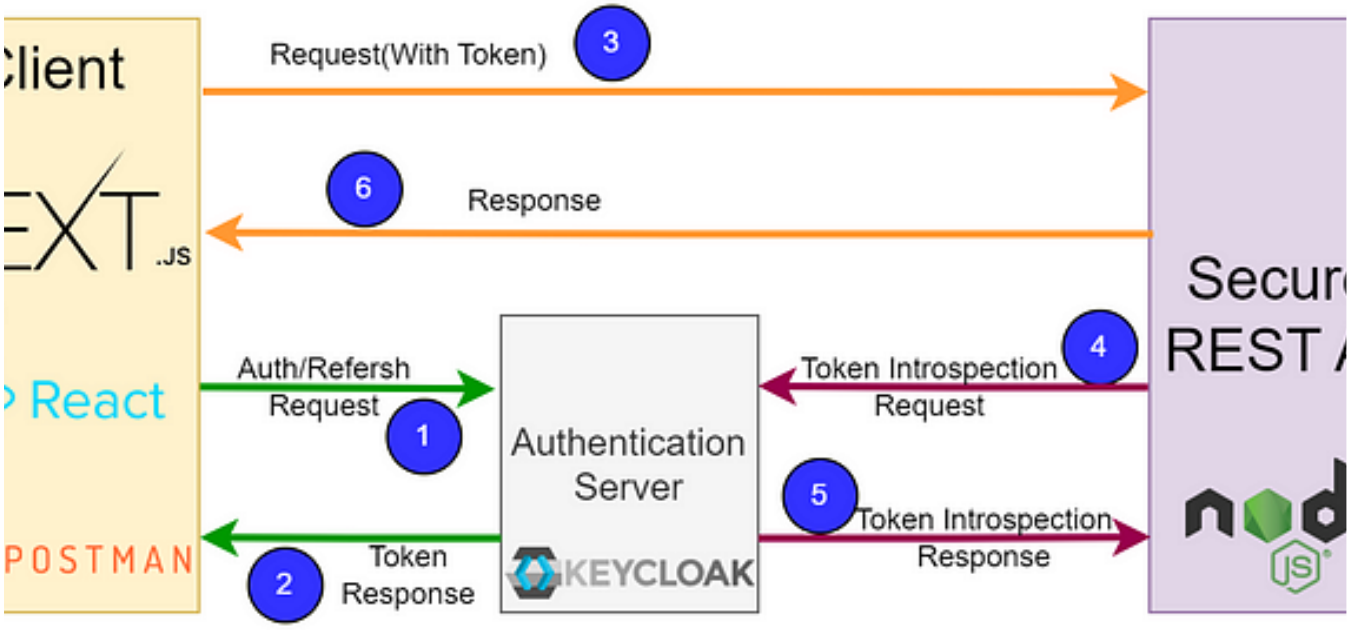
5d ago

Tari Ibaba in Coding Beauty

5 amazing new JavaScript features in ES15 (2024)

5 juicy ES15 features with new functionality for cleaner and shorter JavaScript code in 2024.

Jun 2 2.3K 15



Saurav Samantray

Securing REST API with Role-based access control(RBAC) using Keycloak—Part I (NodeJS)

Learn how to set up Keycloak and integrate it with a Nodejs(Express) application to secure your REST endpoint with role-based access...

May 29 2

See more recommendations