Comprehensive Guide for the User Management API Project

---

1. Introduction

This project is a RESTful API built with **Express.js**, a powerful and lightweight Node.js framework. The primary goal of this API is to manage user data through a modular and maintainable structure. It uses **middleware**, **controllers**, **routes**, and **services** to handle operations like creating, reading, updating, and deleting users (CRUD).

The project is designed with **scalability** and **readability** in mind, making it suitable for developers of all experience levels. By following this guide, you will learn how to:

- Set up and run the project.
- Understand the modular structure and the responsibilities of each part.
- Extend the project with new features while maintaining its consistency.

---

2. Project Setup

2.1 Prerequisites

Before starting, ensure that you have the following installed on your system:

- **Node.js** (version 16 or higher): Download Node.js
- **npm** (comes with Node.js)

You will also need:

- A code editor, preferably **Visual Studio Code**.
- A tool to test API endpoints, such as **Postman** or **Insomnia**.

2.2 Steps to Set Up the Project

Clone the Repository Use `git` to clone the project to your local machine:
bash
Copiar código
```
git clone https://github.com/your-username/your-repository.git
cd your-repository
```

1.

Install Dependencies Install the required Node.js packages:
bash
Copiar código
```
npm install
```

2.

Run the Server Start the application:
bash
Copiar código
```
npm start
```

3. By default, the server will run on:
    - **URL**: `http://localhost:3000`
4. **Test the API** Use **Postman**, **curl**, or another API client to send requests to the endpoints.

---

3. Code Structure

The project follows a modular structure, which separates concerns to enhance maintainability and scalability. Here's an overview of the folder organization:

plaintext
Copiar código

```
.
├── usersGestion/            # Parent folder for user-related logic
│   ├── middlewares/         # Middlewares for user management
│   │   ├── checkUserId.js   # Middleware for validating user ID
│   │   └── validateUser.js  # Middleware for validating user data
│   ├── controllers/         # Logic for user-related endpoints
│   │   └── userController.js # Controllers for users
│   ├── routes/              # User-related routes
│   │   └── userRoutes.js    # Defines user endpoints
│   └── services/            # Services for handling business logic
│       └── userService.js   # Core service functions for users
├── userMock.js              # Simulated database for testing
├── app.js                   # Main application entry point
├── package.json             # Project dependencies and
configuration
└── README.md                # Documentation for the project
```

3.1 Explanation of Each Part

- **`usersGestion/`**:
  - Contains all logic related to user management, such as middlewares, controllers, and routes.
- **Middlewares**:
  - Responsible for pre-processing requests, such as validating user input or IDs.
- **Controllers**:
  - Handle the business logic for each endpoint. For example, fetching a user by ID or creating a new user.
- **Routes**:
  - Define the HTTP methods and paths for the API. Routes connect incoming requests to the appropriate controllers.
- **Services**:
  - Contain reusable functions for performing business operations, such as searching for a user or adding a new user to the database.

---

4. Middlewares

4.1 What are Middlewares?

Middlewares are functions that execute during the request-response lifecycle in an Express.js application. They:

1. Modify the `req` (request) and `res` (response) objects.
2. Stop the request-response cycle or pass control to the next middleware.

4.2 Practical Example:

Here's a middleware for logging requests:

javascript
Copiar código

```
app.use((req, res, next) => {
  console.info(`[INFO] ${req.method} ${req.path}`);
  next();
});
```

4.3 Middleware Types

- **Global Middleware**: Applied to every route in the application. Example: `express.json()` to parse incoming JSON data.
- **Route-Specific Middleware**: Applied only to specific routes. Example: `checkUserIdMiddleware` to validate user IDs.

---

5. Endpoints

5.1 What are Endpoints?

Endpoints define how the server responds to client requests. They are identified by:

- **HTTP Method**: (GET, POST, PUT, DELETE).
- **Route**: A specific path, such as `/users` or `/users/:id`.

5.2 Example Endpoint

Here's an example of a POST endpoint to create a user:

javascript
Copiar código

```
app.post('/users', validateUser, (req, res) => {
  const { nombre, apellido } = req.body;
  const newUser = { id: Date.now(), nombre, apellido };
  users.push(newUser);
  res.status(201).json({ message: 'User created successfully', user:
newUser });
});
```

5.3 Response Codes

- **200 OK**: The request was successful.
- **201 Created**: A new resource was successfully created.
- **400 Bad Request**: The request data is invalid.
- **404 Not Found**: The requested resource does not exist.

---

6. Exports and Imports

6.1 How Exports and Imports Work in Node.js

Exports: Share functions, objects, or variables from a file. Example:
javascript
Copiar código
```
export const getAllUsers = () => { ... };
```

-

Imports: Use the exported functions or variables in another file. Example:
javascript
Copiar código
```
import { getAllUsers } from './services/userService.js';
```

-

6.2 Practical Example

Here's how the `checkUserIdMiddleware` is imported:

javascript
Copiar código
```
import checkUserIdMiddleware from '../middlewares/checkUserId.js';
```

---

7. Best Practices

7.1 Modular Structure

Organize the code into folders for middlewares, routes, controllers, and services. This ensures:

- **Separation of concerns**.
- **Easier debugging and testing**.

7.2 Use Constants

- Use `const` for variables that don't change.
- Use `let` for variables that need to be reassigned.
- Avoid `var`.

7.3 Error Handling

Implement a global error handler:

javascript
Copiar código

```
app.use((error, req, res, next) => {
  res.status(error.statusCode || 500).json({ error: error.message
});
  console.error(error);
});
```