



Buenos
Aires
Ciudad

Agencia de Habilidades
para el Futuro

<Talento
Tech/>

Node.JS

Clase N° 10: Modelando una API Rest

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase.

Índice



1



2

Modelando una API Rest

- API Rest
- Estructura de archivos
- Capas de la aplicación
- División de responsabilidades

REQUEST & RESPONSE

- Rutas
- CORS: solicitudes entre dominios
- Error Handle (404)
- Rutas parametrizadas
- POSTMAN

Objetivos de la Clase

1 Comprender los fundamentos de una API Rest

Poder aplicarlo al proyecto final

2 Estructura de archivos

Veremos como organizar el código de manera eficiente

3 División de responsabilidades

Fomentando el uso de principios de diseño que aseguren un desarrollo más ordenado y modular.

API Rest



¿Qué es una API Rest?

Es una arquitectura o metodología de software que propone la comunicación entre sistemas mediante HTTP y JSON.

Recursos

Trabaja con datos accesibles mediante URLs únicas.

Métodos

Usa GET, POST, PUT y DELETE para interactuar con recursos.

Características Clave de REST



Sin Estado

Cada interacción es **independiente**, facilitando el manejo de múltiples solicitudes.



Formato JSON

Datos enviados en formato fácil de leer para computadoras y desarrolladores.



Interoperabilidad

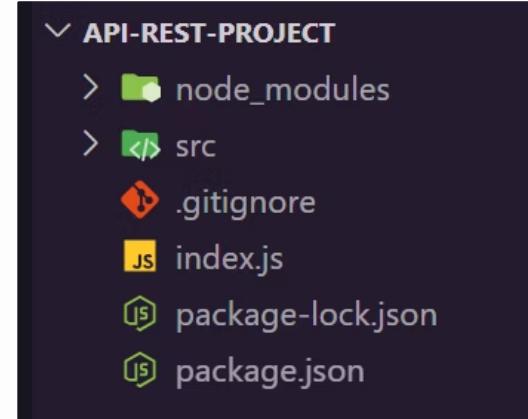
Permite que sistemas con diferentes tecnologías se comuniquen eficazmente.

Estructura de archivos

Importancia de la Estructura

En el desarrollo de software es importante tomar algunas decisiones antes de comenzar a trabajar en el código. Una buena estructura debe adaptarse al tamaño del proyecto. Si es algo pequeño, podemos empezar con algo básico, como un par de carpetas y archivos.

En esta oportunidad aprenderemos una estructura basada en capas, simple y escalable para API Rest desarrolladas en Node junto con Express.



Organización

Facilita encontrar y mantener el código.

Escalabilidad

Permite crecer el proyecto de manera ordenada.

Colaboración

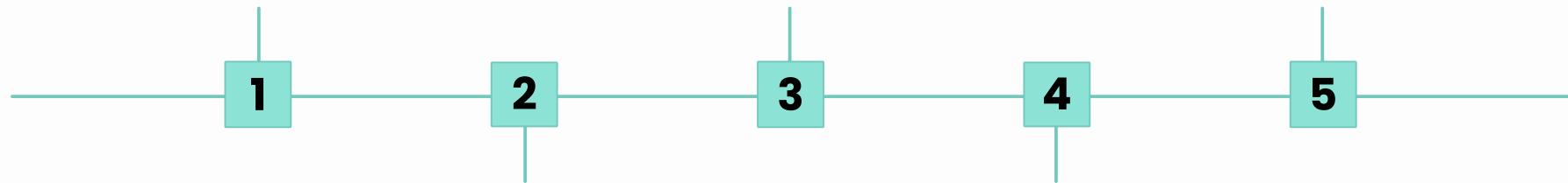
Mejora el trabajo en equipo al tener roles claros.

Estructura de Archivos

src
Carpeta principal para archivos
de la aplicación.

controllers
Manejo de solicitudes y
respuestas.

services
Lógica de negocio principal.



routes
Configuración de rutas de la API.

models
Definición de datos y conexión
con base de datos.

Capas de la aplicación

Capas de la aplicación



Cuando hablamos de las capas de una aplicación, nos referimos a cómo dividimos las responsabilidades dentro del sistema para que cada parte se encargue de algo específico.

Ejemplo:

Es como construir una casa: cada piso tiene un propósito diferente, y eso hace que todo funcione de manera más ordenada y eficiente.

- ⓘ En el caso de una API Rest, dividirla en capas nos ayuda a manejar mejor el código, hacer cambios sin romper todo y trabajar en equipo más fácilmente.

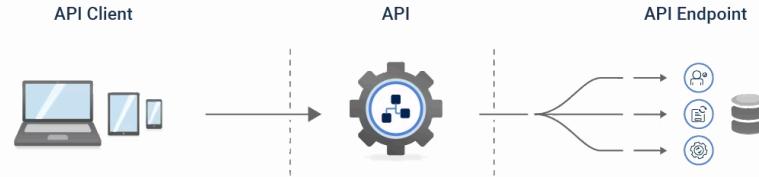
Rutas

Cuando hablamos de rutas en una API Rest, nos referimos a los "puntos de entrada" (también conocidos como **endpoints**) que los clientes (como aplicaciones o navegadores) usan para interactuar con nuestra aplicación. Cada ruta es como una dirección que lleva a un recurso específico dentro de la API y define qué se puede hacer con ese recurso.

Por ejemplo: podrías tener una ruta para listar todos los usuarios, otra para crear uno nuevo y otra para eliminarlo.



Capas de la aplicación: Rutas



Las rutas se construyen combinando una URL específica con un método HTTP. Los métodos más comunes son:

- **GET**, para obtener información (como listar usuarios).
- **POST**, para crear algo nuevo (como registrar un usuario).
- **PUT** o **PATCH**, para actualizar información (como cambiar el correo de un usuario).
- **DELETE**, para eliminar recursos (como borrar un usuario).

Organización

Archivos específicos dentro de una carpeta llamada **routes** o similar.

Capas de la Aplicación: Controladores

Son los responsables de manejar la lógica que conecta las rutas con el resto de la aplicación, actuando como un puente entre las solicitudes del cliente y los datos o servicios que necesitamos usar.

⚠️ Un detalle importante es que los controladores también se encargan las respuestas del servidor y de manejar los errores que puedan surgir.

- ⓘ Imagina que la API es un restaurante: las rutas son los meseros que toman los pedidos, y los controladores son los chefs que preparan y entregan los platos lo que se pide. Cada controlador se enfoca en una tarea específica, lo que ayuda a mantener el código organizado y fácil de entender.

1 Reciben Solicitudes

Procesan las peticiones que llegan desde las rutas.

2 Coordinan Acciones

Deciden qué hacer con cada solicitud.

3 Manejan Respuestas

Devuelven la información procesada al cliente.

Ejemplo de Controlador

Si tienes una API que maneja usuarios, podrías tener un controlador llamado **users.controller.js** con funciones como estas:

- **getAllUsers(req, res)**: Maneja la solicitud para obtener una lista de usuarios y devuelve una respuesta con esa información.
- **getUserById(req, res)**: Busca un usuario específico basado en un parámetro como el id que llega en la solicitud.
- **createUser(req, res)**: Procesa los datos enviados en la solicitud para crear un nuevo usuario.
- **updateUser(req, res)**: Recibe datos actualizados y los aplica a un usuario existente.
- **deleteUser(req, res)**: Se encarga de eliminar un usuario específico.

Capas de la Aplicación: Modelos

En una API, los modelos son como los planos de una máquina: describen la estructura de los datos y su interacción con la base de datos.

Funciones

Definen propiedades, tipos de datos y validaciones para cada entidad.

Los modelos no solo definen la estructura de los datos, sino que también nos facilitan el acceso y manipulación de información almacenada en bases de datos, ya sean relacionales (SQL) o no relacionales (NoSQL).



Ejemplo de Modelo en MongoDB

Si usamos MongoDB con la librería Mongoose, un modelo de usuario podría verse así:

```
const userSchema = new mongoose.Schema({  
    name: { type: String, required: true },  
    email: { type: String, required: true, unique: true },  
    password: { type: String, required: true },  
    createdAt: { type: Date, default: Date.now }  
});
```

*Nota: en nuestro caso no utilizaremos bases de datos tradicionales por lo que no será necesario definir modelos de datos como estrictamente los conocemos, Por lo que la capa de modelos nos servirá para interactuar con la información.

Capas de la Aplicación: Servicios

Los servicios son una parte clave de la arquitectura de una API Rest y actúan como los “motores” que realizan las operaciones necesarias para cumplir con las solicitudes que llegan a la API. Su función principal es manejar la lógica de negocio de la aplicación, es decir, todas las reglas y procesos que determinan cómo los datos deben ser tratados.

¿Cómo funcionan los servicios?

1 Desde el controlador

Reciben las solicitudes para ejecutar tareas específicas.

2 Hacia los modelos

Envían solicitudes para obtener o modificar datos en la base de datos, o en otros casos, procesan datos externos.

Ejemplo de servicios

```
const productModel = require('./product.model');
async function getProductsInStock() {
    const products = await productModel.getAllProducts();
    return products.filter(product => product.stock > 0);
} async function addNewProduct(productData) {
    if (!productData.name || !productData.price) {
        throw new Error('El nombre y el precio del producto son obligatorios.');
    }
}
```

```
}

if (productData.price <= 0) {
    throw new Error('El precio debe ser mayor que cero.');
}

return await productModel.createProduct(productData);
}

module.exports = {
    getProductsInStock,
    addNewProduct,
};
```

División de responsabilidades

División de Responsabilidades

1 Planificación estructurada:

Diseña la estructura de archivos y directorios antes de comenzar.

2 División de responsabilidades:

Cada capa de la API debe tener un propósito específico. Evita concentrar todo el código en un único archivo.

3 Uso de módulos:

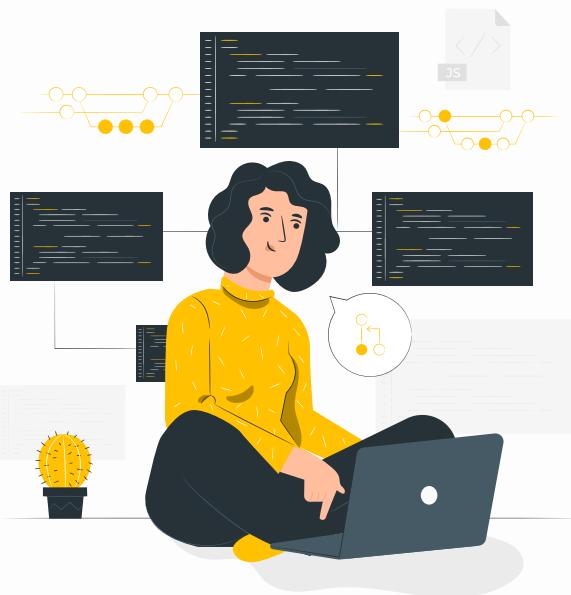
Exporta e importa partes del código de forma ordenada. Garantiza un control granular sobre su uso.

Beneficios

- Aplicaciones fáciles de mantener.
- Mayor escalabilidad y evolución eficiente.



Una arquitectura modular y planificada asegura proyectos organizados, robustos y listos para el futuro.



Beneficios de la Arquitectura en Capas



Modularidad

Facilita el desarrollo y prueba de componentes por separado.



Reutilización

Permite usar el mismo código en diferentes partes de la aplicación.



Flexibilidad

Hace más fácil adaptar la aplicación a nuevos requisitos.

Conclusiones y Próximos Pasos

1 Práctica

Aplica estos conceptos en tu próximo proyecto de API.

2 Profundización

Explora temas como Request y Response, y trabajo con JSON.

3 Evolución

Mantente actualizado con las mejores prácticas en desarrollo de APIs.

Materiales y Recursos Adicionales

1 Guía de Arquitectura REST:

Un artículo que detalla los principios básicos de la arquitectura REST, que es la base de la organización de nuestra API.

2 Tutorial sobre la creación de APIs RESTful con Express y Node.js:

Este tutorial paso a paso te llevará desde la creación de un servidor básico hasta la implementación de rutas, controladores, y modelos.



Ejercicio Práctico



Ejercicio Práctico

Obligatorio

Estructura y Rutas de la API

Ahora que ya tienes una buena base trabajando con APIs, es momento de ponerte en los zapatos de un desarrollador que construye una aplicación desde cero:



"Construir una API no solo se trata de hacer que los datos lleguen a tu aplicación, sino de cómo organizas todo el proyecto de manera que sea fácil de mantener y escalar", explica Sabrina mientras te señala la pantalla.



"Queremos que crees la estructura de directorios para tu API usando arquitectura REST", dice Matías con tono serio, "Esto significa que debes dividir tu código de manera ordenada en capas, tal como lo vimos en la clase. Queremos ver que uses una estructura clara para manejar rutas, controladores, modelos y servicios. Así, cuando tu proyecto crezca, será más fácil mantenerlo organizado y profesional."



Ejercicio Práctico

Obligatorio

Misión:

1. Crea la estructura de directorios para tu aplicación, asegurándote de crear carpetas para **rutas, controladores, modelos y servicios**.

2. En tu archivo principal, crea dos rutas nuevas:

- Una ruta que devuelva una respuesta en formato **HTML**. Podría ser algo simple, como una página de bienvenida.
- Otra ruta que devuelva una respuesta en formato **JSON**, por ejemplo, con una lista de usuarios o productos ficticios.
- Todavía no es necesario colocar las rutas dentro de la carpeta routes, por lo que deberás definirlas en el archivo index.js.



¡Nuevo cuestionario en Campus!

- ⓘ No olvides que los cuestionarios son de carácter obligatorio para poder avanzar con la cursada.

