



Agencia de Habilidades
para el Futuro

<Talento
Tech/>

Node.JS

Clase N° 12 - Capa lógica

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase.

Índice



1



2

Capa lógica

- Express Router
- Controladores
- Servicios
- Arquitectura basada en capas

Modelo de datos y trabajo con JSON

- **JSON**
 - Parse
 - Stringify
- **Filesystem**
- **Modelos**

Objetivos de la Clase

Express Router

Modularizar las rutas de la aplicación para una gestión más clara y ordenada.

Controladores

Separar la lógica de recepción de solicitudes y coordinación de respuestas.

Servicios

Encapsular la lógica de negocio reutilizable, promoviendo un código limpio y modular.

Arquitectura en capas

Comprender cómo esta estructura facilita el manejo de la lógica de la aplicación.

- ⓘ Aprenderemos a estructurar aplicaciones web robustas con Node.js y Express, enfocándonos en la organización del código para un mantenimiento y escalabilidad eficientes.

Express Router

¿Por qué usar Express Router? ¿Cómo funciona Express Router?

Express Router

¿Qué es?

Un middleware de Express.js que permite crear manejadores de rutas modulares y montables.

Función

Agrupa rutas en diferentes archivos o módulos, facilitando la escalabilidad y organización del código.

Ventajas de Express Router

Organización

Facilita la organización del código al separar las rutas en módulos lógicos.

Reutilización

Permite reutilizar conjuntos de rutas en diferentes partes de la aplicación.

Mantenibilidad

Mejora la mantenibilidad al aislar la lógica de cada conjunto de rutas.

Legibilidad

Aumenta la legibilidad del código al dividirlo en partes más pequeñas y manejables.

¿Cómo funciona Express Router?

Así se ve el archivo index.js de nuestro proyecto:

```
import express from 'express';
import cors from 'cors';

const app = express();

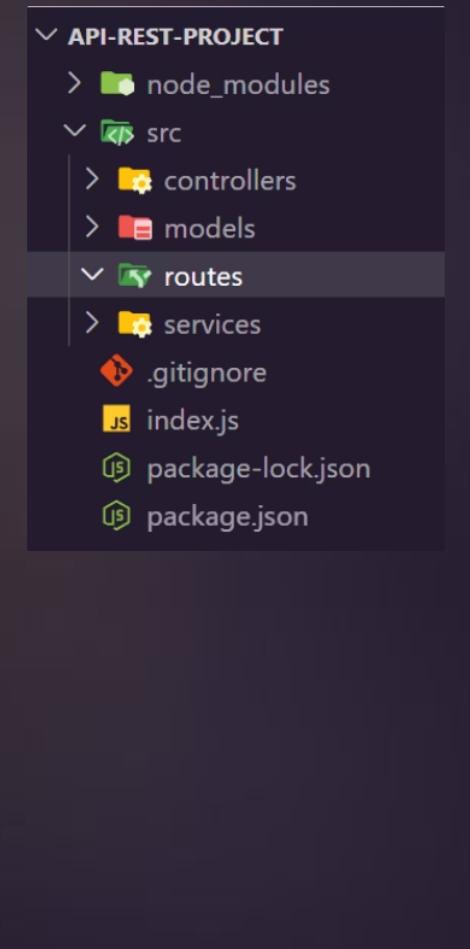
// Configuración básica: Permitir todos los orígenes
app.use(cors());

// Aquí irían las rutas

// Middleware para manejar errores 404
app.use((req, res, next) => {
    res.status(404).send('Recurso no encontrado');
});

const PORT = 3000;
app.listen(PORT, () => console.log(`http://localhost:${PORT}`));
```

Implementación de Express Router



Crear archivo de rutas

Agrupar rutas por modelo de negocio en archivos separados.

Definir rutas en el Router

Usar métodos como `get`, `post`, `put`, `delete` en la instancia de router.

1

2

3

4

Crear instancia de Router

Utilizar `express.Router()` en el archivo de rutas.

Invocar el Router

Utilizar `app.use()` en el archivo principal para integrar el Router.

Estructura de archivos con Express Router

index.js

Configuración inicial del servidor y uso de routers.

products.routes.js

Definición de rutas específicas para productos.

Ejemplo de Express Router

```
// products.routes.js
import express from 'express';
const router = express.Router();

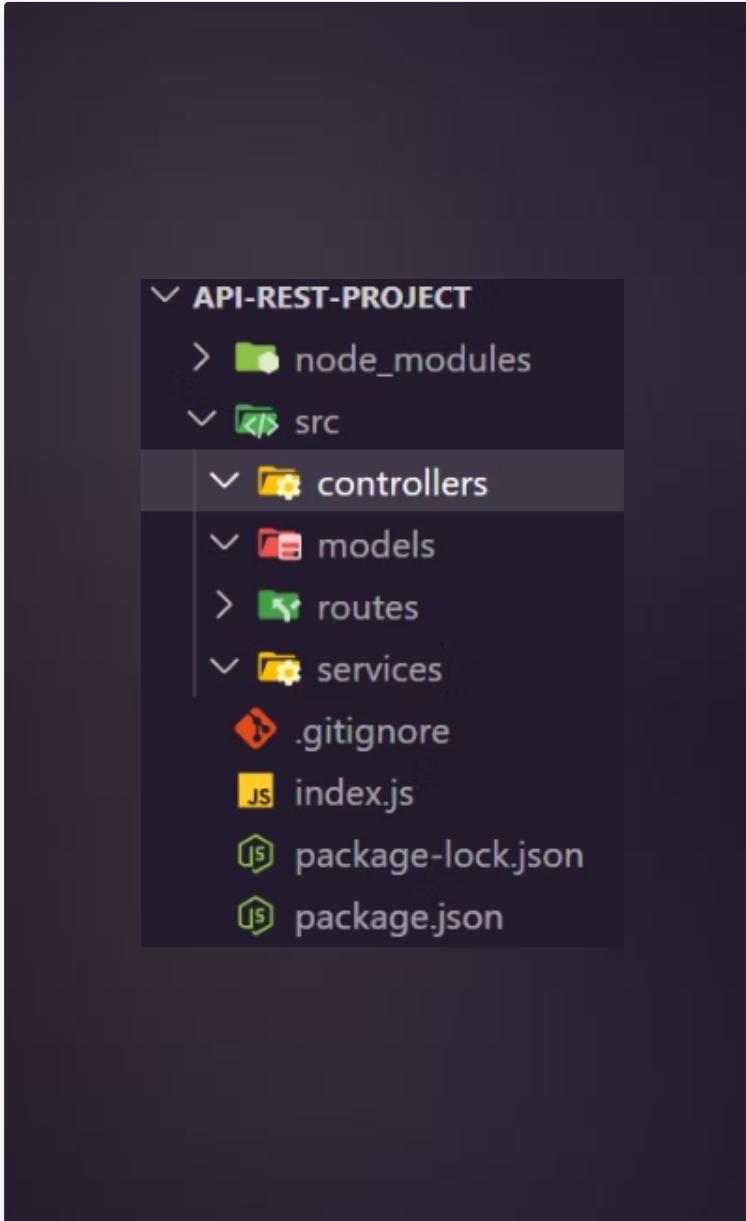
router.get('/products', (req, res) => {
  res.send('Listado de productos');
});

export default router;

// index.js
import productsRouter from
'./routes/products.routes.js';
app.use('/api', productsRouter);
```

Controladores

¿Cuál es la función de un Controlador? Creando un controlador



Controladores en Express

Capa que gestiona las solicitudes HTTP que llegan a los diferentes endpoints.

Función

Interactúa con la lógica de negocio y genera las respuestas correspondientes.

Funciones de un Controlador

Recibir solicitudes

Procesa la solicitud HTTP del cliente desde una ruta.

Generar una respuesta

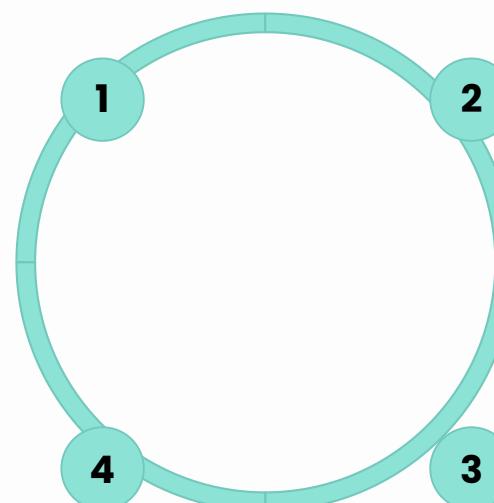
Crea y envía una respuesta HTTP al cliente.

Procesar la solicitud

Extrae información relevante y realiza validaciones necesarias.

Interactuar con el modelo/servicio

Realiza operaciones necesarias con los datos o lógica de negocio.



Creando un Controlador



Crear archivo

Crear products.controller.js en la carpeta controllers.

1

Definir funciones

Implementar funciones para cada operación CRUD.

2

Asociar con rutas

Vincular funciones del controlador con las rutas correspondientes.

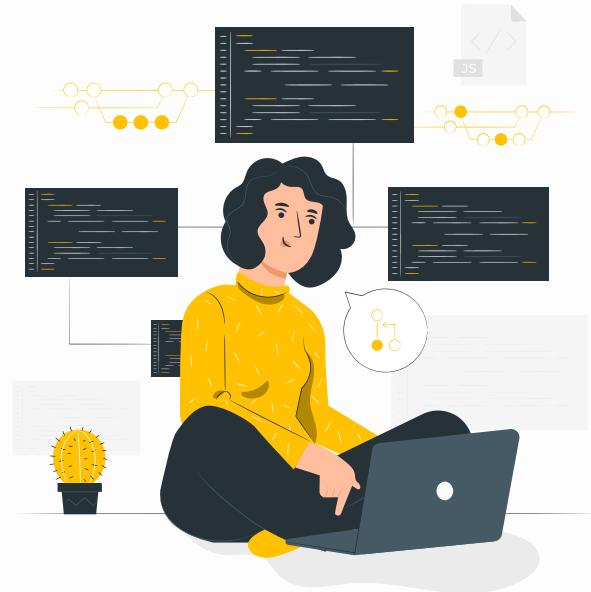
3

Ejemplo de Controlador

```
// products.controller.js

export const getAllProducts = async (req, res) => {
  res.status(200).json(products);
};

export const getProductById = async (req, res) => {
  const id = req.params.id;
  const product = products.find(product =>
product.id == id);
  if (product) {
    res.status(200).json(product);
  } else {
    res.status(404).json({ message: 'Producto no
encontrado' });
  }
};
```



Servicios

¿Cuál es la función de un Servicio? Creando un Servicio

Servicios

Capa que encapsula la lógica de negocio de la aplicación.

Función

Actúa como intermediario entre controladores y modelos o fuentes de datos.

Funciones de un Servicio

Encapsular lógica de negocio

Contiene reglas y procesos específicos de la aplicación.

Abstraer acceso a datos

Proporciona una interfaz limpia para operaciones de datos.

Reutilización de lógica

Permite usar la misma lógica en diferentes partes de la aplicación.

Creando un Servicio

Ejemplo de Servicio

1

Crear archivo

Crear products.service.js en la carpeta services.

2

Definir funciones

Implementar funciones para manipulación de datos y lógica de negocio.

3

Exportar funciones

Hacer disponibles las funciones para su uso en controladores.

```
// products.service.js
const products = [
  { id: 1, name: 'Producto 1', price: 1000 },
  { id: 2, name: 'Producto 2', price: 2000 }
];

export const getAllProducts = () => {
  return products;
};

export const getProductById = async (id) => {
  return products.find(product =>
  product.id == id);
};
```

Arquitectura basada en capas

Arquitectura basada en capas



Rutas

Define los endpoints de la API



Controladores

Maneja solicitudes y respuestas



Servicios

Encapsula lógica de negocio



Modelos/Datos

Representa y gestiona los datos

Ventajas de la arquitectura en capas

Separación de responsabilidades

Cada capa tiene una función específica y bien definida.

Modularidad

Facilita la organización y mantenimiento del código.

Escalabilidad

Permite crecer y modificar la aplicación con mayor facilidad.

Reutilización

Promueve la reutilización de código en diferentes partes de la aplicación.

Flexibilidad en la arquitectura

Adaptabilidad

La arquitectura puede adaptarse según las necesidades del proyecto.

Evolución

Con la práctica, se perfecciona la aplicación de estos conceptos.

Recursos Adicionales

Documentación Oficial de Express

Explora en detalle cómo configurar rutas, middlewares y manejar errores.

Documentación sobre Express Router

Sección específica dedicada al Router en Express.js.



Preguntas para Reflexionar

Express Router

¿Cuáles son las ventajas de usar Express Router vs definir todas las rutas en la aplicación principal?

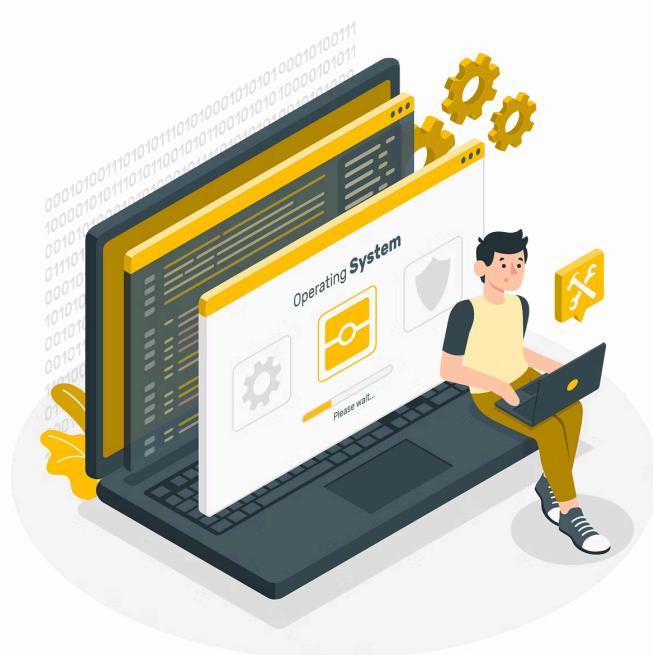
Controladores

¿Cuál es la principal responsabilidad de un controlador en una API REST?

Servicios

¿Qué ventajas ofrece usar servicios en cuanto a la reutilización de código?





Próximos Pasos

- 1** — **Modelo de datos y JSON**
Consultar datos internos y devolverlos al cliente desde la API Rest.
- 2** — **Datos en la nube**
Configurar y acceder a datos en un servidor externo.
- 3** — **Autenticación y Autorización**
Manejar el acceso público y privado de los datos.



Ejercicio Práctico



Ejercicio Práctico

Obligatorio

Organización de Rutas y Capas en tu API

Sabrina y Matías se acercan a revisar tu progreso. Matías sonríe con aprobación: "Has avanzado muchísimo, pero ahora es momento de subir un nivel. Necesitamos que configures la capa de rutas de manera profesional."



Matías, con una expresión confiada, te dice: "Es hora de que trabajes como un verdadero desarrollador profesional. Sabemos que manejar rutas en el archivo principal está bien para comenzar, pero si tu aplicación crece, esa organización será insostenible."

Sabrina da un paso adelante y añade: "Por eso, queremos que trabajes con **Express Router** para separar tus rutas en módulos, que crees controladores para manejar la lógica y que comiences a trabajar con una capa de servicios para preparar datos simulados que luego enviarás a través de tus rutas."



Ejercicio Práctico

Obligatorio

Matías asiente y explica: "Este enfoque no solo hará que tu código sea más limpio, sino que también será más fácil de mantener y escalar. ¡Manos a la obra!"

Misión:

1. Crear rutas organizadas con Express Router
 - Migra las rutas de tu archivo principal a archivos separados en una carpeta llamada **routes** por ejemplo: **products.routes.js**
 - Usa **Express Router** para configurar el o los archivos de rutas y asegúrate de exportarlos correctamente para que pueda ser utilizado en el archivo principal.

2. Implementar controladores para manejar la lógica
 - Crea un archivo llamado **product.controller.js** dentro de la carpeta **controllers**.
 - Crea los controladores necesarios para responder a las rutas definidas en el ejercicio anterior.
 - Mueve la lógica de las rutas al controlador correspondiente y asegúrate de que las funciones sean claras y reutilizables.



Ejercicio Práctico

Obligatorio

-
3. Añadir una capa de servicios con datos simulados

- Crea un archivo llamado `product.service.js` dentro de la carpeta `services`.
- Simula datos en formato JSON, como una lista de productos o usuarios, y utiliza estas funciones en los controladores para devolver respuestas dinámicas.

Ejemplo práctico:

Supongamos que tienes una ruta que devuelve una lista de productos. Este es el flujo que debes implementar:

- **Archivo `productRoutes.js` (en la carpeta `routes`)**
Define las rutas principales y vincúlalas a las funciones del controlador.
- **Archivo `productController.js` (en la carpeta `controllers`)**
Implementa la lógica de cada ruta, como obtener productos o filtrar por categoría.
- **Archivo `productService.js` (en la carpeta `services`)**
Crea funciones que devuelvan datos simulados para que puedan ser utilizados en el controlador.



Ejercicio Práctico

Obligatorio



Cuando termines, no olvides probar todo con **POSTMAN**. Es la herramienta ideal para asegurarte de que las rutas, controladores y servicios estén funcionando perfectamente.

Este ejercicio es crucial para consolidar lo que has aprendido. Cuando termines, tendrás una base sólida para cualquier proyecto que emprendas. ¡Buena suerte!



Tu desafío está claro. Ahora es momento de estructurar, modularizar y organizar como un profesional. ¡El código te espera!



¡Nuevo cuestionario en Campus!

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

