



**DATOS DEL ESTUDIANTE**

Apellidos y Nombres:	TASAYCO HUACCAMAYTA DIEGO ALONSO	ID:	1552480
Dirección Zonal/CFP:	CFP CHINCHA		
Carrera:	INGENIERIA DE SOFTWARE CON INTELIGENCIA ARTIFICIAL	Semestre:	V
Curso/ Mód. Formativo:	FULLSTACK DEVELOPER SOFTWARE		
Tema de Trabajo Final:	TAREA 08		

**1. INFORMACIÓN****▪ Identifica la problemática del caso práctico propuesto.**

Gestión manual e ineficiente de la información del personal de una empresa, dificultando el registro, actualización y eliminación de datos de los empleados. La ausencia de un sistema automatizado genera errores humanos, pérdida de información y demora en los procesos administrativos.

**▪ Identifica propuesta de solución y evidencias.**

Desarrollo de un sistema backend en Node.js con Express y MySQL que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los empleados.

El sistema facilita la administración de los datos de forma centralizada y eficiente mediante endpoints probados con Thunder Client.

Se implementó una estructura modular basada en controladores, rutas y configuración de base de datos, asegurando escalabilidad y mantenimiento del código.

- **Respuestas a preguntas guía**

Durante el análisis y estudio del caso práctico, debes obtener las respuestas a las interrogantes:

Pregunta 01:	
Pregunta 02:	
Pregunta 03:	
Pregunta 04:	
Pregunta 05:	

## 2. PLANIFICACIÓN DEL TRABAJO

### ▪ Cronograma de actividades:

N°	ACTIVIDADES	CRONOGRAMA					
1							
2							
3							
4							
5							
6							

### ▪ Lista de recursos necesarios:

1. MÁQUINAS Y EQUIPOS	
Descripción	Cantidad
ADM Ryzen 5 5600GT with Radeon Graphics	1
Memoria Ram 16 GB	1
Placa base: B450M PRO-VDH MAX	1

2. HERRAMIENTAS E INSTRUMENTOS	
Descripción	Cantidad
Bloc de notas	1
Word	1
Visual Studio Code	1

3. MATERIALES E INSUMOS	
Descripción	Cantidad
Git Hub	1

### 3. DECIDIR PROPUESTA

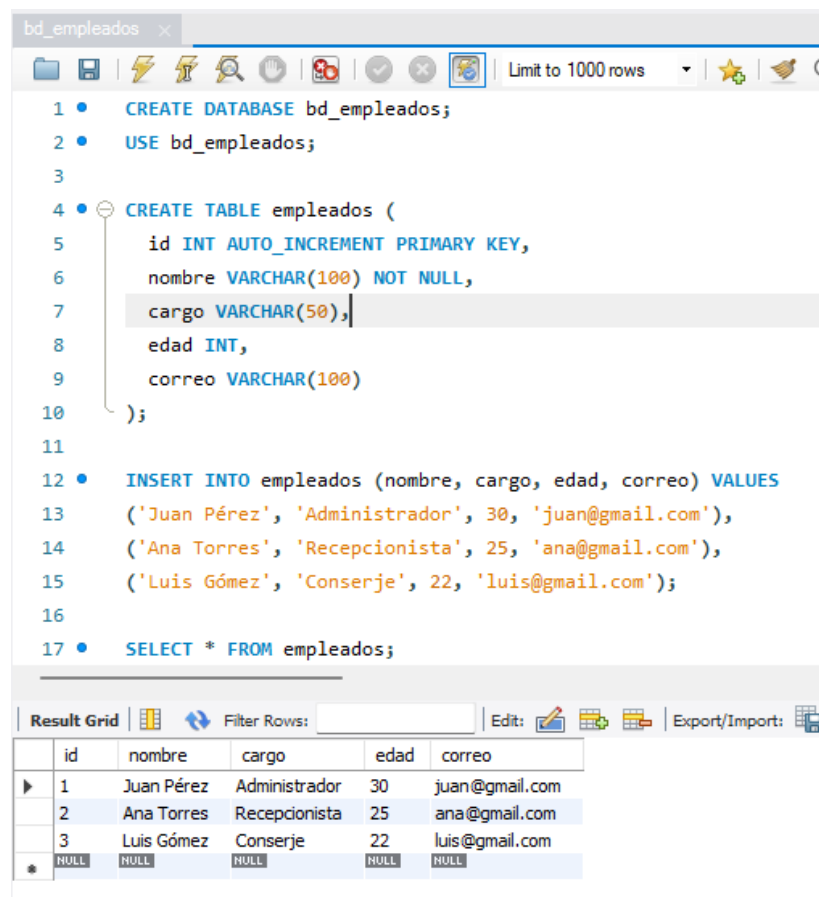
- Describe la propuesta determinada para la solución del caso práctico

#### PROPUESTA DE SOLUCIÓN

## TAREA 08

Implementación del CRUD de Empleados en Node.js (Backend) y mysql  
PROCEDIMIENTOS:

La base de datos se llama **bd\_empleados** y contiene una tabla principal llamada **empleados**, que almacena la información básica de cada trabajador. Esta base de datos se conecta al backend desarrollado.



```

1 • CREATE DATABASE bd_empleados;
2 • USE bd_empleados;
3
4 • CREATE TABLE empleados (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     nombre VARCHAR(100) NOT NULL,
7     cargo VARCHAR(50),
8     edad INT,
9     correo VARCHAR(100)
10 );
11
12 • INSERT INTO empleados (nombre, cargo, edad, correo) VALUES
13 ('Juan Pérez', 'Administrador', 30, 'juan@gmail.com'),
14 ('Ana Torres', 'Recepcionista', 25, 'ana@gmail.com'),
15 ('Luis Gómez', 'Conserje', 22, 'luis@gmail.com');
16
17 • SELECT * FROM empleados;
  
```

	id	nombre	cargo	edad	correo
▶	1	Juan Pérez	Administrador	30	juan@gmail.com
	2	Ana Torres	Recepcionista	25	ana@gmail.com
	3	Luis Gómez	Conserje	22	luis@gmail.com
*	NULL	NULL	NULL	NULL	NULL

## 4. EJECUTAR

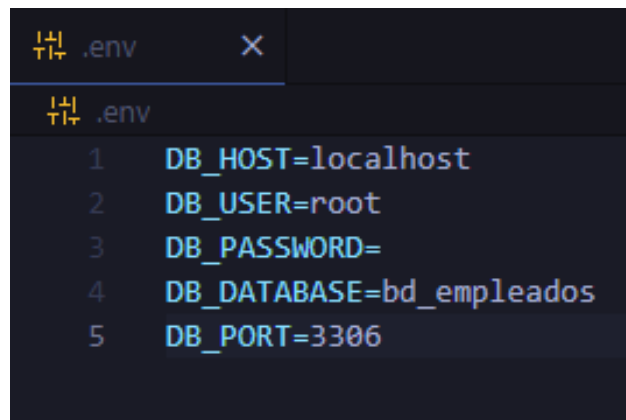
- Resolver el caso práctico, utilizando como referencia el problema propuesto y las preguntas guía proporcionadas para orientar el desarrollo.
- Fundamentar sus propuestas en los conocimientos adquiridos a lo largo del curso, aplicando lo aprendido en las tareas y operaciones descritas en los contenidos curriculares.

**INSTRUCCIONES:** Ser lo más explícito posible. Los gráficos ayudan a transmitir mejor las ideas. Tomar en cuenta los aspectos de calidad, medio ambiente y SHI.

OPERACIONES / PASOS / SUBPASOS	NORMAS TÉCNICAS - ESTANDARES / SEGURIDAD / MEDIO AMBIENTE
Crear la base de datos bd_empleados y la tabla empleados en MySQL.	
Configurar el archivo db.js con los datos de conexión a MySQL.	
Crear el archivo server.js para levantar el servidor con Express.	
Definir las rutas del CRUD en empleadoRoutes.js	
Implementar la lógica de las operaciones en empleadoController.js.	
Probar los endpoints (GET, POST, PUT, DELETE) con Thunder Client.	
Verificar el almacenamiento, actualización y eliminación de registros en la base de datos.	
Documentar el proyecto con un archivo README.md.	

## BACKEND:

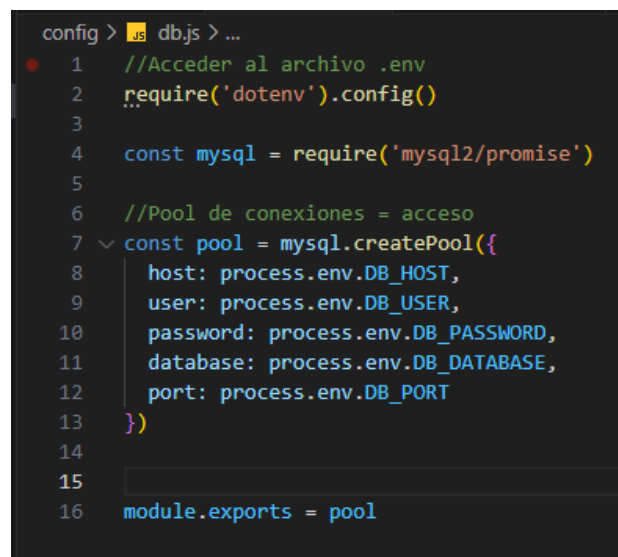
Creación del archivo .env para definir las variables de entorno del proyecto, incluyendo la configuración de la base de datos (host, usuario, contraseña, nombre de la BD y puerto del servidor).



```
.env
1 DB_HOST=localhost
2 DB_USER=root
3 DB_PASSWORD=
4 DB_DATABASE=bd_empleados
5 DB_PORT=3306
```

### Config/db.js:

Esto define la conexión a la base de datos sin esto el controlador no podrá ejecutar consultas.



```
config > JS db.js > ...
1 //Acceder al archivo .env
2 require('dotenv').config()
3
4 const mysql = require('mysql2/promise')
5
6 //Pool de conexiones = acceso
7 const pool = mysql.createPool({
8   host: process.env.DB_HOST,
9   user: process.env.DB_USER,
10  password: process.env.DB_PASSWORD,
11  database: process.env.DB_DATABASE,
12  port: process.env.DB_PORT
13 })
14
15
16 module.exports = pool
```

**Controllers/empleadoController:**

Usa la conexión de db.js para hacer las operaciones (crear, leer, actualizar, eliminar).  
**(Listar, crear)**

```

controllers > ls empleadoController.js > ...
1  const db = require('../config/db')
2
3  // Listar
4  const obtenerTodos = async (req, res) => {
5    try {
6      const [rows] = await db.query('SELECT * FROM empleados')
7      res.json(rows)
8    } catch (e) {
9      res.status(500).json({ error: 'Error interno en el servidor' })
10   }
11 }
12
13 // Crear
14 const crear = async (req, res) => {
15   try {
16     const { nombre, cargo, edad, correo } = req.body
17     if (!nombre || !cargo || !edad || !correo)
18       return res.status(400).json({ error: 'Todos los campos son obligatorios' })
19
20     const [result] = await db.query(
21       'INSERT INTO empleados (nombre, cargo, edad, correo) VALUES (?, ?, ?, ?)',
22       [nombre, cargo, edad, correo]
23     )
24     res.status(201).json({ id: result.insertId, message: 'Empleado creado correctamente' })
25   } catch (e) {
26     res.status(500).json({ error: 'Error al crear empleado' })
27   }
28 }

```

**(Actualizar, Eliminar)**

```

controllers > ls empleadoController.js > ...
30 // Actualizar
31 const actualizar = async (req, res) => {
32   try {
33     const { id } = req.params
34     const { nombre, cargo, edad, correo } = req.body
35     const [result] = await db.query(
36       'UPDATE empleados SET nombre=?, cargo=?, edad=?, correo=? WHERE id=?',
37       [nombre, cargo, edad, correo, id]
38     )
39     if (result.affectedRows === 0) return res.status(404).json({ error: 'Empleado no encontrado' })
40     res.json({ message: 'Empleado actualizado correctamente' })
41   } catch (e) {
42     res.status(500).json({ error: 'Error al actualizar empleado' })
43   }
44 }
45
46 // Eliminar
47 const eliminar = async (req, res) => {
48   try {
49     const { id } = req.params
50     const [result] = await db.query('DELETE FROM empleados WHERE id=?', [id])
51     if (result.affectedRows === 0) return res.status(404).json({ error: 'Empleado no encontrado' })
52     res.json({ message: 'Empleado eliminado correctamente' })
53   } catch (e) {
54     res.status(500).json({ error: 'Error al eliminar empleado' })
55   }
56 }
57
58 module.exports = { obtenerTodos, crear, actualizar, eliminar }

```



**-routes/empleadoRoutes.js**

Las rutas de la API llaman a las funciones del controlador.

```
server.js db.js empleadoController.js empleadoRoutes.js X TC
routes > empleadoRoutes.js > ...
1  const express = require('express')
2  const router = express.Router()
3  const empleadoController = require('../controllers/empleadoController')
4
5  // CRUD
6  router.get('/', empleadoController.obtenerTodos) // GET - listar
7  router.post('/', empleadoController.crear) // POST - crear
8  router.put('/:id', empleadoController.actualizar) // PUT - actualizar
9  router.delete('/:id', empleadoController.eliminar) // DELETE - eliminar
10
11 module.exports = router
12
```

-server.js: Configura el servidor y define las rutas base de la API.

```
server.js > ...
1  const express = require('express')
2  const cors = require('cors')
3
4  // Rutas
5  const empleadoRoutes = require('./routes/empleadoRoutes')
6
7  const app = express()
8  const PORT = process.env.PORT || 3000
9
10 // CONFIGURACIÓN
11 // Permisos CORS
12 app.use(cors({
13   origin: '*',
14   methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
15   credentials: true
16 }))
17 // JSON
18 app.use(express.json())
19
20 // Rutas API
21 app.use('/api/empleados', empleadoRoutes)
22
23 // Iniciar servidor
24 app.listen(PORT, () => {
25   console.log(`Servidor iniciado en http://localhost:${PORT}`)
26 })
27
```

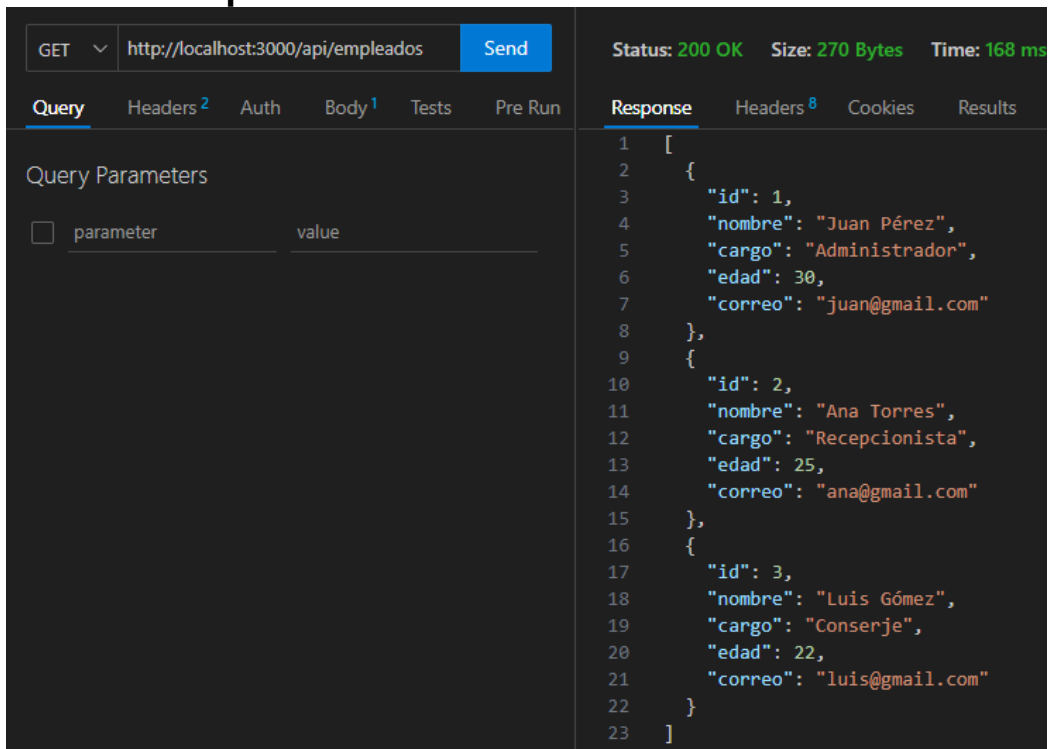
## EJECUCIÓN DEL SERVIDOR:

Una vez iniciado el servidor con el comando:

“nodemon server”

-Se realizan las pruebas de los cuatro métodos principales del CRUD (**GET**, **POST**, **PUT** y **DELETE**) utilizando la herramienta **Thunder Client**.

### 1. GET - Listar empleados



GET ⌵ http://localhost:3000/api/empleados Send Status: 200 OK Size: 270 Bytes Time: 168 ms

Query Headers<sup>2</sup> Auth Body<sup>1</sup> Tests Pre Run

Query Parameters

☐ parameter value

Response Headers<sup>8</sup> Cookies Results

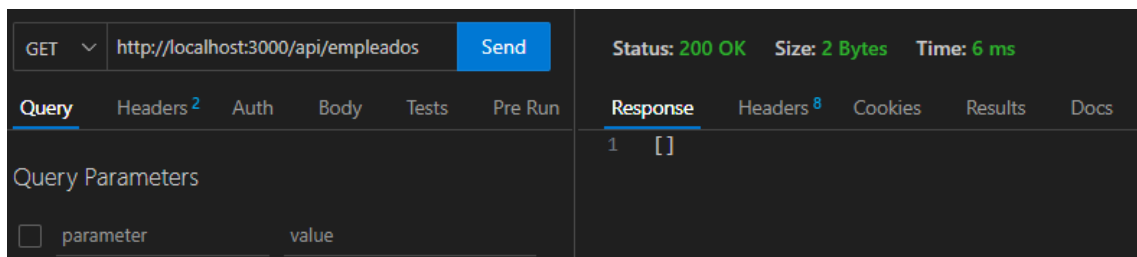
```

1  [
2    {
3      "id": 1,
4      "nombre": "Juan Pérez",
5      "cargo": "Administrador",
6      "edad": 30,
7      "correo": "juan@gmail.com"
8    },
9    {
10     "id": 2,
11     "nombre": "Ana Torres",
12     "cargo": "Recepcionista",
13     "edad": 25,
14     "correo": "ana@gmail.com"
15   },
16   {
17     "id": 3,
18     "nombre": "Luis Gómez",
19     "cargo": "Conserje",
20     "edad": 22,
21     "correo": "luis@gmail.com"
22   }
23 ]

```

### Prueba de error:

Si la base de datos está vacía, simplemente devuelve un arreglo vacío .



GET ⌵ http://localhost:3000/api/empleados Send Status: 200 OK Size: 2 Bytes Time: 6 ms

Query Headers<sup>2</sup> Auth Body Tests Pre Run

Query Parameters

☐ parameter value

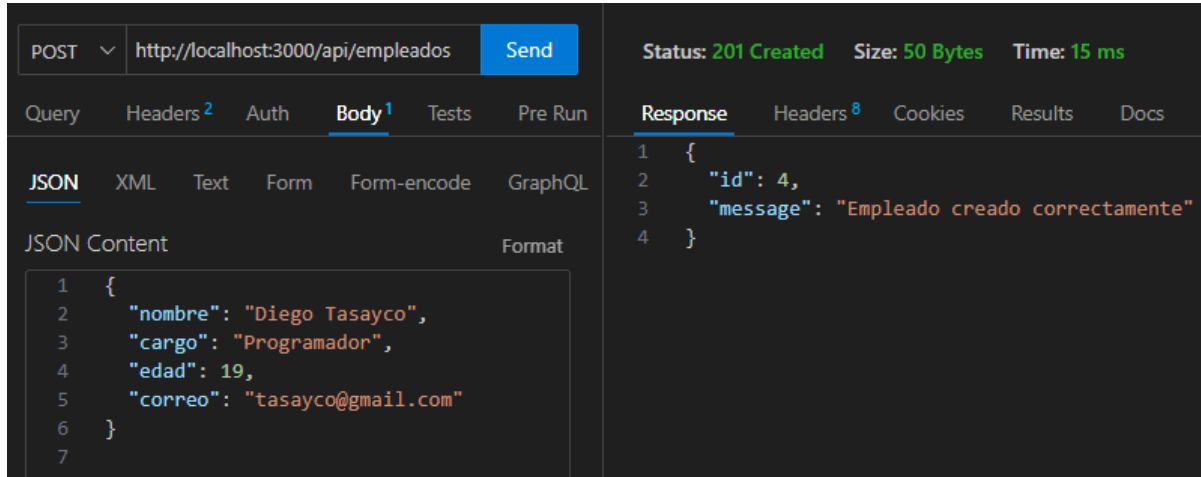
Response Headers<sup>8</sup> Cookies Results Docs

```

1  []

```

## 2. POST - Crear empleado



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/empleados
- Status:** 201 Created
- Size:** 50 Bytes
- Time:** 15 ms
- Body (Request):**

```

1 {
2   "nombre": "Diego Tasayco",
3   "cargo": "Programador",
4   "edad": 19,
5   "correo": "tasayco@gmail.com"
6 }
7

```
- Response:**

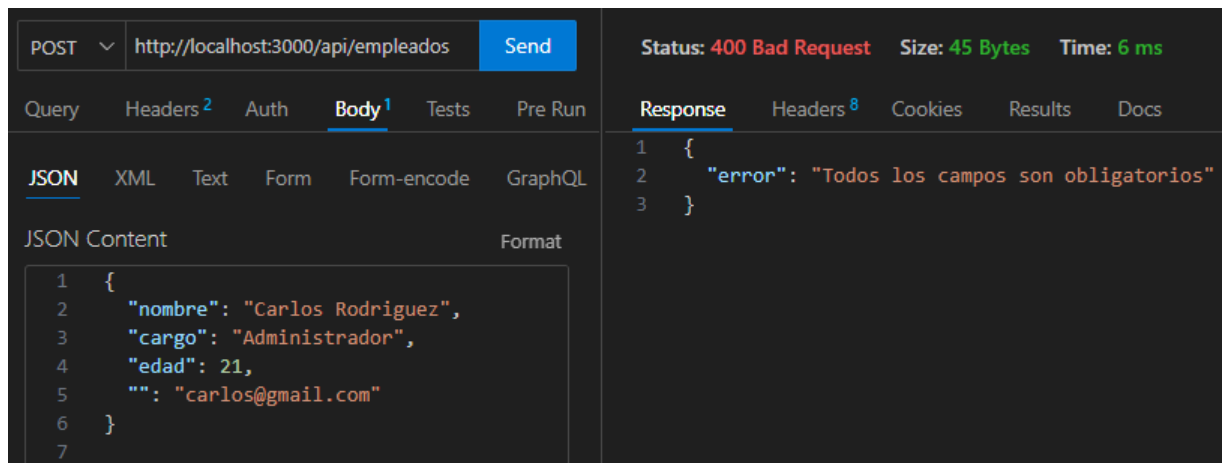
```

1 {
2   "id": 4,
3   "message": "Empleado creado correctamente"
4 }

```

### Prueba de error:

Si falta algún campo, el sistema devuelve:



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/empleados
- Status:** 400 Bad Request
- Size:** 45 Bytes
- Time:** 6 ms
- Body (Request):**

```

1 {
2   "nombre": "Carlos Rodriguez",
3   "cargo": "Administrador",
4   "edad": 21,
5   "": "carlos@gmail.com"
6 }
7

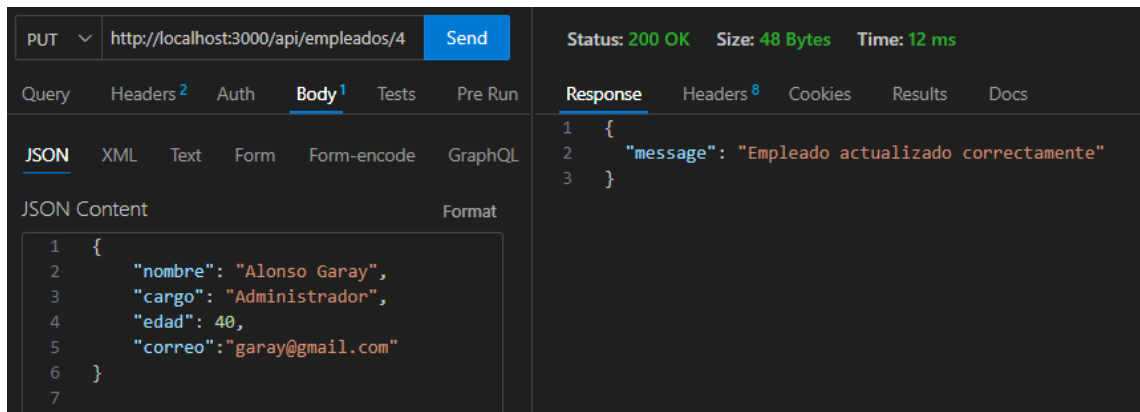
```
- Response:**

```

1 {
2   "error": "Todos los campos son obligatorios"
3 }

```

### 3. PUT - Actualizar empleado



The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:3000/api/empleados/4
- Status:** 200 OK
- Size:** 48 Bytes
- Time:** 12 ms
- Body (Request):**

```

1 {
2   "nombre": "Alonso Garay",
3   "cargo": "Administrador",
4   "edad": 40,
5   "correo": "garay@gmail.com"
6 }
7

```
- Response (JSON):**

```

1 {
2   "message": "Empleado actualizado correctamente"
3 }

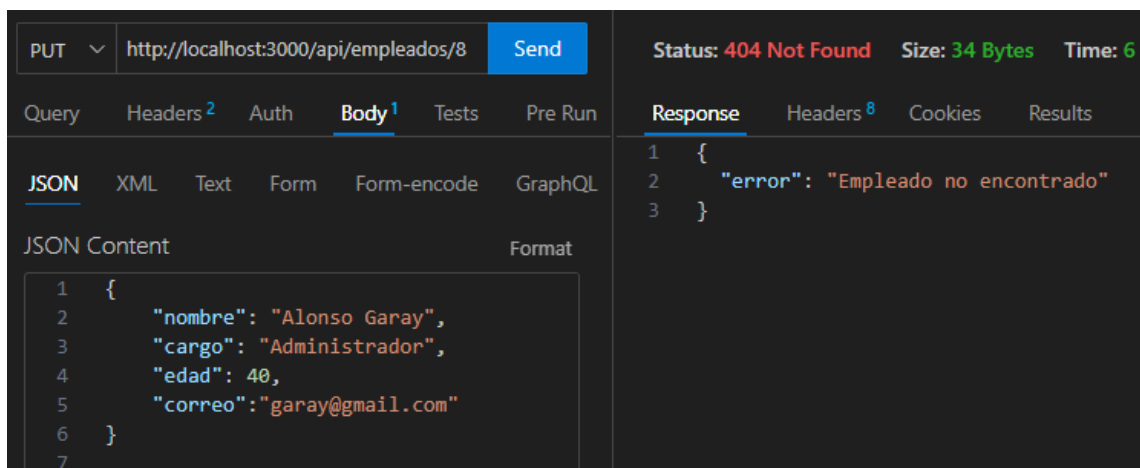
```

```

10   "id": 2,
11   "nombre": "Ana Torres",
12   "cargo": "Recepcionista",
13   "edad": 25,
14   "correo": "ana@gmail.com"
15 },
16 {
17   "id": 3,
18   "nombre": "Luis Gómez",
19   "cargo": "Conserje",
20   "edad": 22,
21   "correo": "luis@gmail.com"
22 },
23 {
24   "id": 4,
25   "nombre": "Alonso Garay",
26   "cargo": "Administrador",
27   "edad": 40,
28   "correo": "garay@gmail.com"
29 }

```

**Prueba de error:**  
Si el ID no existe:



The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:3000/api/empleados/8
- Status:** 404 Not Found
- Size:** 34 Bytes
- Time:** 6 ms
- Body (Request):**

```

1 {
2   "nombre": "Alonso Garay",
3   "cargo": "Administrador",
4   "edad": 40,
5   "correo": "garay@gmail.com"
6 }
7

```
- Response (JSON):**

```

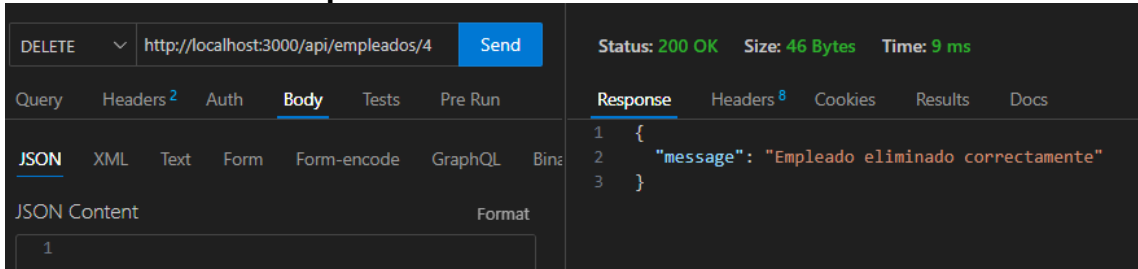
1 {
2   "error": "Empleado no encontrado"
3 }

```

## DIBUJO / ESQUEMA / DIAGRAMA DE PROPUESTA

(Adicionar las páginas que sean necesarias)

### 4. DELETE - Eliminar empleado



DELETE ▼ http://localhost:3000/api/empleados/4 Send

Status: **200 OK** Size: **46 Bytes** Time: **9 ms**

Query Headers<sup>2</sup> Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Bin

JSON Content Format

1

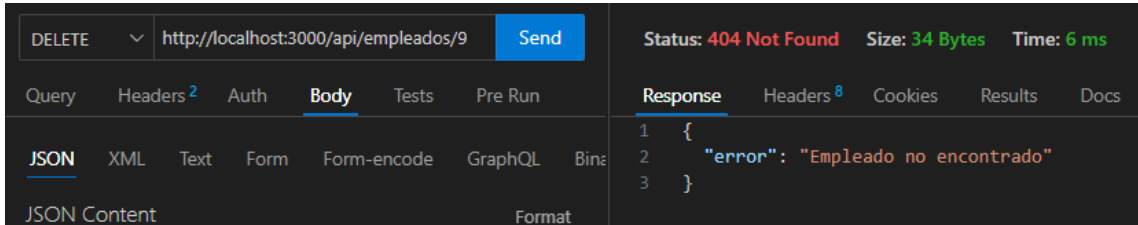
Response Headers<sup>8</sup> Cookies Results Docs

```

1 {
2   "message": "Empleado eliminado correctamente"
3 }
```

### Prueba de error:

Si se intenta eliminar un ID inexistente:



DELETE ▼ http://localhost:3000/api/empleados/9 Send

Status: **404 Not Found** Size: **34 Bytes** Time: **6 ms**

Query Headers<sup>2</sup> Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Bin

JSON Content Format

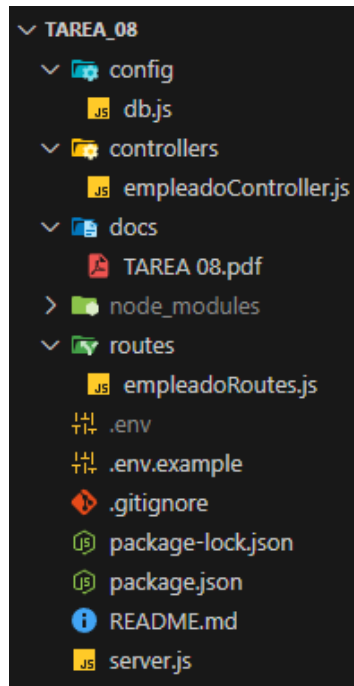
1

Response Headers<sup>8</sup> Cookies Results Docs

```

1 {
2   "error": "Empleado no encontrado"
3 }
```

### Carpetas creadas:



- ▼ TAREA\_08
  - ▼ config
    - db.js
  - ▼ controllers
    - empleadoController.js
  - ▼ docs
    - TAREA 08.pdf
  - > node\_modules
  - ▼ routes
    - empleadoRoutes.js
  - .env
  - .env.example
  - .gitignore
  - package-lock.json
  - package.json
  - README.md
  - server.js

## 5. CONTROLAR

- **Verificar el cumplimiento de los procesos desarrollados en la propuesta de solución del caso práctico.**

EVIDENCIAS	CUMPLE	NO CUMPLE
• ¿Se identificó claramente la problemática del caso práctico?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se desarrolló las condiciones de los requerimientos solicitados?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se formularon respuestas claras y fundamentadas a todas las preguntas guía?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se elaboró un cronograma claro de actividades a ejecutar?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se identificaron y listaron los recursos (máquinas, equipos, herramientas, materiales) necesarios para ejecutar la propuesta?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se ejecutó la propuesta de acuerdo con la planificación y cronograma establecidos?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se describieron todas las operaciones y pasos seguidos para garantizar la correcta ejecución?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se consideran las normativas técnicas, de seguridad y medio ambiente en la propuesta de solución?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
• ¿La propuesta es pertinente con los requerimientos solicitados?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• ¿Se evaluó la viabilidad de la propuesta para un contexto real?	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## 6. VALORAR

- Califica el impacto que representa la propuesta de solución ante la situación planteada en el caso práctico.

CRITERIO DE EVALUACIÓN	DESCRIPCIÓN DEL CRITERIO	PUNTUACIÓN MÁXIMA	PUNTAJE CALIFICADO POR EL ESTUDIANTE
Identificación del problema	Claridad en la identificación del problema planteado.	3	
Relevancia de la propuesta de solución	La propuesta responde adecuadamente al problema planteado y es relevante para el contexto del caso práctico.	8	
Viabilidad técnica	La solución es técnicamente factible, tomando en cuenta los recursos y conocimientos disponibles.	6	
Cumplimiento de Normas	La solución cumple con todas las normas técnicas de seguridad, higiene y medio ambiente.	3	
<b>PUNTAJE TOTAL</b>		<b>20</b>	

