Módulo 5.1: Backend - Node/Express

MSc. Carlos Cedeño

Introducción al Backend

El **backend** es la parte de una aplicación web que se ejecuta en el servidor y es responsable de la lógica del negocio, el manejo de datos y la comunicación con la base de datos y otros servicios. El usuario final no interactúa directamente con el backend, pero este es esencial para que la aplicación funcione correctamente.

¿Qué es la lógica de negocio en una aplicación informática?

La lógica de negocio en el backend es el conjunto de reglas, cálculos y procesos personalizados que manejan los datos para que una aplicación funcione como se espera. Es el cerebro de la aplicación que toma las decisiones y ejecuta las tareas específicas del negocio.

En pocas palabras, es el código que implementa las reglas de operación de tu empresa o proyecto.

La lógica de negocio es el intermediario entre la interfaz de usuario (frontend) y la base de datos. Se encarga de:

- **Procesar datos**: Recibe información del usuario (como un formulario de registro) y la procesa.
- **Validar y aplicar reglas**: Se asegura de que los datos cumplan con ciertas condiciones.
- **Realizar cálculos**: Calcula totales, impuestos, descuentos o cualquier otra operación matemática necesaria.
- **Orquestar flujos de trabajo**: Ejecuta una secuencia de pasos. Por ejemplo, al realizar una compra, la lógica de negocio debe verificar el stock del producto.

¿Qué es Node.js?

Node.js es un entorno de ejecución de JavaScript del lado del servidor. Permite a los desarrolladores usar JavaScript para escribir el código del backend, lo que ofrece la ventaja de utilizar un solo lenguaje en todo el stack de desarrollo.

¿Qué es Express?

Express es un framework minimalista y flexible para aplicaciones web de Node.js. Proporciona un conjunto robusto de características para desarrollar aplicaciones web y APIs de forma sencilla.

Iniciar Proyecto con Node.js y Express

Prerrequisitos

Asegúrate de tener instalado **Node.js** y **npm**. Puedes descargarlos desde nodejs.org.

Tip para la vida: Usar NVM (Node version manager) para olvidarnos de futuros problemas.

Inicialización del Proyecto

1. Crea una nueva carpeta para tu proyecto y navega hacia ella en tu terminal.
2. Inicia un nuevo proyecto de Node.js con el siguiente comando. Esto creará un archivo `package.json`.

```
npm init -y
```

3. Instala Express:

```
npm install express
```

Creando tu Primer Servidor

Crea un archivo llamado `index.js` y añade el siguiente código:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('¡Hola, mundo!');
});

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

Para ejecutar tu servidor, corre el siguiente comando en la terminal:

```
node index.js
```

Ahora, si abres tu navegador y vas a `http://localhost:3000`, verás el mensaje "¡Hola, mundo!".

📂 Estructura de un Proyecto en Express

```
/mi-proyecto
|-- /node_modules
|-- /src
|   |-- /controllers
|   |   `-- userController.js
|   |-- /models
|   |   `-- userModel.js
|   |-- /routes
|   |   `-- userRoutes.js
|   |-- /config
|   |   `-- firebase.js
|   `-- index.js
|-- .env
|-- .gitignore
`-- package.json
```

- **`src/`** : Contiene todo el código fuente de la aplicación.
- **`controllers/`** : Manejan la lógica de negocio. Reciben las peticiones, procesan los datos (a menudo interactuando con los modelos) y envían una respuesta.
- **`models/`** : Definen la estructura de los datos y la interacción con la base de datos.
- **`routes/`** : Definen las rutas de tu API y las asocian con los controladores correspondientes.
- **`config/`** : Almacena los archivos de configuración, como la conexión a la base de datos.
- **`.env`** : Archivo para guardar variables de entorno (credenciales de la base de datos, claves de API, etc.).
- **`.gitignore`** : Especifica los archivos y carpetas que Git debe ignorar (como `node_modules` y `.env`).

🔥 Conectando tu Proyecto a Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web de Google que ofrece una variedad de herramientas y servicios, incluyendo bases de datos en tiempo real (Realtime Database y Firestore), autenticación, hosting y más.

1. Crear un Proyecto en Firebase

1. Ve a la Consola de Firebase.
2. Haz clic en **"Añadir proyecto"** y sigue los pasos para crear un nuevo proyecto.
3. Una vez creado, serás redirigido al panel principal de tu proyecto.

2. Configurar el SDK de Firebase Admin

Para conectar tu servidor de Node.js con Firebase, necesitas el **SDK de Firebase Admin**.

1. En el panel de tu proyecto de Firebase, ve a **"Configuración del proyecto"** (el ícono de engranaje).
2. Selecciona la pestaña **"Cuentas de servicio"**.
3. Haz clic en **"Generar nueva clave privada"**. Se descargará un archivo JSON con tus credenciales. **¡Guarda este archivo de forma segura y no lo compartas públicamente!**

3. Integrar Firebase en tu Proyecto de Node.js

1. **Instala el paquete `firebase-admin`:**

   ```
   npm install firebase-admin
   ```

2. Mueve el archivo JSON que descargaste a tu proyecto, preferiblemente dentro de una carpeta `config`.

3. Crea un archivo en `src/config/firebase.js` para inicializar Firebase:

```
const admin = require('firebase-admin');
const serviceAccount = require('./tu-archivo-de-credenciales.json'); // Asegúrate de que la ruta sea correcta

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
  databaseURL: "REEMPLAZA POR EL URL DE TU BD"
});

const db = admin.database();

module.exports = { db };
```

Implementando el CRUD

Ahora, vamos a crear el controlador y las rutas para gestionar "items" en nuestra base de datos.

1. Controlador del CRUD (`itemController.js`)

`src/controllers/itemController.js`

```
const { db } = require('../config/firebase');

// Referencia a la colección "items" en la base de datos
const itemsRef = db.ref('items');

// CREATE: Añadir un nuevo item
const createItem = async (req, res) => {
  try {
    const newItemRef = itemsRef.push(); // Genera un ID único
    await newItemRef.set(req.body);
    res.status(201).json({ id: newItemRef.key, ...req.body });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// READ: Obtener todos los items
const getItems = async (req, res) => {
  try {
    const snapshot = await itemsRef.once('value');
    res.status(200).json(snapshot.val());
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// UPDATE: Actualizar un item por su ID
const updateItem = async (req, res) => {
  try {
    const { id } = req.params;
    await itemsRef.child(id).update(req.body);
    res.status(200).json({ id, ...req.body });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// DELETE: Borrar un item por su ID
const deleteItem = async (req, res) => {
  try {
    const { id } = req.params;
    await itemsRef.child(id).remove();
    res.status(200).json({ message: `Item ${id} eliminado correctamente.` });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

module.exports = {
  createItem,
  getItems,
  updateItem,
```

2. Rutas del CRUD (`itemRoutes.js`)

`src/routes/itemRoutes.js`

```
const express = require('express');
const router = express.Router();
const {
  createItem,
  getItems,
  updateItem,
  deleteItem,
} = require('../controllers/itemController');

// Ruta para crear un nuevo item
router.post('/items', createItem);

// Ruta para obtener todos los items
router.get('/items', getItems);

// Ruta para actualizar un item por su ID
router.put('/items/:id', updateItem);

// Ruta para borrar un item por su ID
router.delete('/items/:id', deleteItem);
```

3. Servidor Principal (`index.js`)

`src/index.js`

```
const express = require('express');
const itemRoutes = require('./routes/itemRoutes');

const app = express();
const port = 3000;

// Middleware para que Express pueda entender JSON en el cuerpo de las peticiones
app.use(express.json());

// Usar las rutas del CRUD bajo el prefijo /api
app.use('/api', itemRoutes);

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

✅ Probando la API

Para probar tu API, puedes usar herramientas como **Postman** o **curl**.

- **Crear un item (POST):**
  - URL: `http://localhost:3000/api/items`
  - Body (JSON): `{ "nombre": "Laptop", "precio": 1200 }`
- **Obtener todos los items (GET):**
  - URL: `http://localhost:3000/api/items`

- **Actualizar un item (PUT):**
  - URL: `http://localhost:3000/api/items/-Nq...` (reemplaza con un ID real)
  - Body (JSON): `{ "precio": 1150 }`
- **Borrar un item (DELETE):**
  - URL: `http://localhost:3000/api/items/-Nq...` (reemplaza con un ID real)