Construcción aplicación web

Ojo iniciar con:

node app.js

ejecutar el index.html

P3. Actividades de aprendizaje de la competencia 220501096 - Desarrollar la solución de *software* de acuerdo con el diseño y metodologías de desarrollo

Con base en los requisitos de *software* especificados tanto funcionales como no funcionales, los documentos de diseño (diagrama de clases, diagrama E-R, diagrama de componentes, diagramas de actividad, diagramas de secuencia, diagrama de paquetes y de arquitectura del sistema) además de los documentos de casos de uso o historias de usuario, en esta competencia se empieza a codificar en el lenguaje seleccionado los prototipos del modelo necesarios para la puesta en marcha de la solución del sistema.

3.1.1 Actividad de aprendizaje GA8-220501096-AA1- Integrar módulos

Esta actividad se centra en el estudio de los mecanismos de desarrollo de *software* que permiten construir los componentes que conforman o integran los módulos de una aplicación web.

Duración: 96 horas.

Materiales de formación: para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: “Construcción aplicación web.”.

Evidencias:

A continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje:

P5. Introducción a Frontend y Backend

En el desarrollo de software y aplicaciones web, normalmente se divide el trabajo en dos grandes áreas: Frontend y Backend. Cada una cumple funciones diferentes, pero ambas se complementan para ofrecer una experiencia completa al usuario.

En este componente formativo se abordarán los conceptos y técnicas que apoyarán el desarrollo de una aplicación web de manera completa, es decir, se realizará el Backend y Front-end de la aplicación, de esta forma se podrá observar la comunicación, intercambio y procesamiento de la información de estos dos componentes.

P6. La siguiente figura representa el esquema general de comunicación entre la aplicación (Front-end) y el servidor (Backend).

¿Qué es el Frontend?

El Frontend es la parte del software o sitio web que el usuario puede ver y con la que interactúa directamente.

Todo lo visual forma parte del Frontend: botones, menús, colores, imágenes, animaciones y la estructura general de la interfaz.

✦ Tecnologías principales del Frontend

- HTML → estructura del contenido.
- CSS → estilos, diseño y apariencia.
- JavaScript → interacción, animaciones y lógica del navegador.

✦ Objetivo del Frontend

- Hacer que la aplicación sea intuitiva, atractiva y fácil de usar.
- Ofrecer una experiencia rápida y agradable al usuario.

⚙ ¿Qué es el Backend?

El Backend es la parte que no se ve, pero que hace posible el funcionamiento interno de la aplicación.
Aquí se procesan los datos, se aplican reglas de negocio y se conecta con la base de datos o servicios externos.

✦ Qué hace el Backend

- Gestiona peticiones que vienen del frontend.
- Guarda y consulta datos en bases de datos.
- Aplica toda la lógica del sistema.
- Controla seguridad, autenticación y permisos.

✦ Tecnologías comunes del Backend

- Lenguajes como: Java, Python, PHP, JavaScript (Node.js), C#, Ruby.
- Bases de datos como: MySQL, PostgreSQL, MongoDB.
- Frameworks: Spring Boot, Django, Express.js, Laravel, .NET.

🔗 ¿Cómo trabajan juntos?

Se comunican normalmente mediante APIs.
El frontend hace una petición (por ejemplo: "quiero los productos del inventario") y el backend responde con los datos listos para mostrar.

📌 Ejemplo sencillo:

1. El usuario hace clic en "Iniciar sesión".
2. El frontend envía los datos al backend.
3. El backend valida la información y responde si es correcto o no.
4. El frontend muestra el resultado en pantalla.

🧩 En resumen

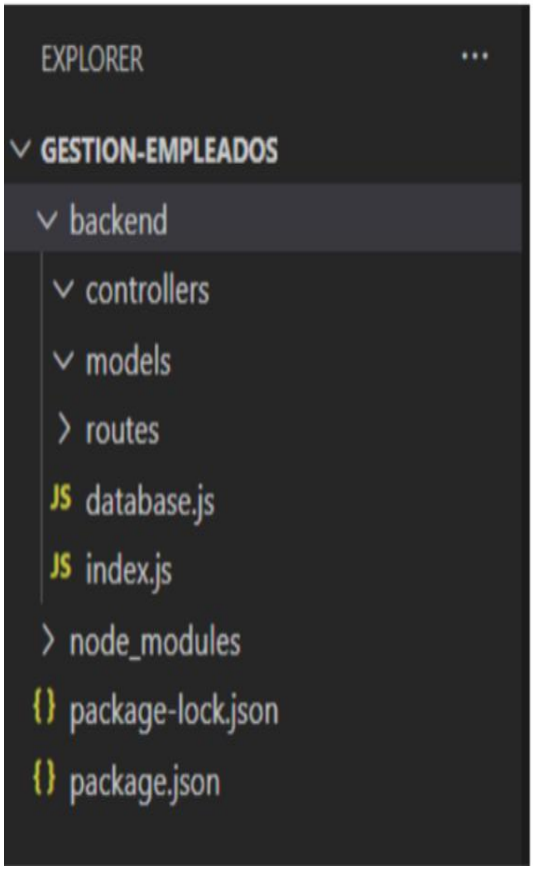
Parte	¿Qué es?	Se encarga de…	Tecnologías
Frontend	**Lo que el usuario ve**	**Interfaz, diseño e interacción**	**HTML, CSS, JS**

Parte	¿Qué es?	Se encarga de…	Tecnologías
Backend	Lo que no se ve	Lógica, datos, seguridad	Java, Python, Node.js, SQL

P7. Construcción del Backend de la aplicación web

Se inicia con una estructura de proyecto MEAN (Mongo, Express, Angular y Node.js) para el proyecto que abarca tanto el Front-end como el Backend. Para la instalación de las primeras herramientas como Node.js y Express en la sección material complementario de este documento encontrará la guía de instalación de Node.js y de Express. A continuación, se incluye una figura que representa de manera general el proceso de codificación de *software*.

1.1 Instalación y configuración de herramientas de Node.js



Se utiliza el Visual Studio Code (VS Code) con la siguiente estructura de directorios que se muestra en la figura 3 para el Backend.

Como se observa en la figura anterior se crea la carpeta GESTION-EMPLEADOS, que será la principal o raíz del proyecto. Dentro de esta se crea la carpeta backend, en la cual se crean otros directorios y archivos como se alcanza a apreciar en la figura. De tal forma, que esta será la estructura que apoyará el desarrollo de la base de la aplicación.

P8. Una vez creadas las carpetas y archivos como lo son *controllers, models, routes*, database.js e index.js, se abre la terminal del VS Code y se digita el siguiente comando para crear un proyecto Node.js, como se muestra en el ejemplo de la siguiente figura:

```
npm init - -yes
```

Una vez creado el proyecto se requiere todas las bondades del *framework* de Node.js, para lo cual se hace necesario Express, que va a generar una serie de componentes que ayudarán mucho al desarrollo de esta aplicación; por lo

tanto, se instala Express con la siguiente línea de comandos desde el terminal del VS Code:

```
npm install express
```

1.2 Desarrollo del index.js

El siguiente paso es iniciar con el desarrollo del archivo principal del Backend, que se llama index.js como se muestra en la siguiente figura, (puede descargar el contenido de estas instrucciones en el archivo index.js disponible en la carpeta de anexos).

P9. En la figura anterior se aprecia el código del archivo index.js para el proyecto, por lo cual en la medida del avance de este componente se irá explicando cada línea de código que se requiera ejecutar. En ese sentido, se inicia con el requerimiento de Express y se almacena en una constante en JavaScript llamada Express. De esta forma se tendrá el acceso a todo el funcionamiento del servidor con la línea de código:

```
const app = express();
```

Una vez creada la constante app, se pasa a la configuración del puerto por donde va a escuchar y a recibir el servidor todo tipo de solicitudes por parte del cliente, según se muestra en la siguiente línea:

```
app.set('port', process.env.PORT || 3000);
```

De este modo, *set* es para crear una variable que va a ser accedida desde cualquier parte de la aplicación, por lo que la estructura es la siguiente (nombre var, valor); *process* es porque cuando se despliegue la app no se va a tener la opción de definir el puerto, sino que el mismo servicio de la nube ayudará a definirlo, en el caso de que el puerto 3000 esté ocupado.

Con el fin de revisar los cambios en el servidor de manera automática se tiene planeada la instalación de algunas herramientas que harán más fácil el desarrollo sobre Node.js, por lo que se instala Nodemon (Vortexbird, 2017) como una utilidad que monitorea de manera constante los cambios en el código fuente que se está desarrollando y de manera automática reinicia el servidor. Por lo tanto, la línea de código utilizada en la terminal de VS Code es:

```
npm install nodemon –D
```

Esto quiere decir que Nodemon será instalado como una dependencia de desarrollo y no como dependencia general del proyecto. Por lo que se configura el archivo package.json según se muestra en la siguiente figura (puede descargar el contenido de estas instrucciones en el archivo package.json disponible en la carpeta), y se inicia a ejecutar el servidor con la siguiente línea de comando:

```
npm run dev
```

Figura 6 *Estructura package.json*

```
{
  "name": "gestion-empleados",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon BackEnd/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongoose": "^5.12.14",
    "Morgan": "^1.10.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
}
```

P10. Ahora, se construyen las Middleware dentro del archivo de index.js como se puede apreciar en la figura anterior, encargadas de procesar los datos, es decir, cuando se solicitan o envían al servidor. Al respecto, el servidor debe entender los datos que le llegan del cliente (AngularJS toma los datos del cliente y los envía en formato JSON) y es por eso que se necesita algún tipo de conversión, por lo que se requiere del uso del formato JSON ejecutando la siguiente instrucción:

```
app.use(express.json());
```

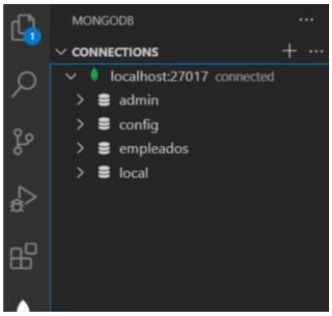
Con el fin de observar la transacción de mensajes por consola se tiene una herramienta útil llamada *morgan*, que ayuda a mirar en consola lo que el usuario está solicitando. Esta herramienta es opcional para el desarrollo de la aplicación, no obstante se deja la línea de comando para su instalación y utilización:

```
npm install Morgan
```

```
app.use(morgan('dev'));
```

1.3 Configuración e instalación de la base de datos MongoDB

A continuación, se realizará la conexión de la aplicación del servidor a la base de datos, por lo que se inicia a configurar y a instalar los recursos necesarios antes de iniciar a desarrollar. En ese sentido, se instala MongoDB de manera persistente en el equipo de desarrollo, para lo cual se deja información para su instalación en la sección material complementario de este documento.



De igual manera, a nivel opcional se instala la extensión de MongoDB dentro del entorno de VS Code, permitiendo tener una integralidad para el desarrollo, estas herramientas facilitan la celeridad en el proceso de construcción de sistemas de *software*. Por lo tanto, en el material de apoyo se dejará información para acceder a la guía para la instalación de esta extensión, denominada extensión VS Code y MongoDB. La siguiente figura muestra de qué manera se visualiza dicha extensión dentro del entorno de desarrollo.

P11. Posteriormente, y con el fin de realizar la conexión a la base de datos y definir la estructura de los datos de la aplicación a desarrollar, o lo que técnicamente se conoce como esquemas, se instala la herramienta Mongoose con la siguiente línea de comando desde la terminal de VS Code:

```
npm install mongoose
```

1.4 Configuración de database.js

Con todas las herramientas necesarias para la conexión de la aplicación a la base de datos se empieza a desarrollar y a configurar sobre el archivo database.js. (Puede descargar el contenido de estas instrucciones en el archivo database.js disponible en la carpeta de anexos).

```
const mongoose = require('mongoose');
  const URI = 'mongodb://localhost/empleados';
  mongoose.connect(URI)
    .then(db => console.log('DB is connected'))
    .catch(err => console.error(err));
module.exports = mongoose;
```

Con respecto al código del archivo database.js, allí se indica que es necesario que el módulo se conecte a MongoDB, por eso aparece Mongoose. Así, al llamarse al módulo se almacena en una constante, por otra parte, se define la dirección de la conexión de la base de datos en una constante denominada URI y, se utiliza una promesa para obtener la información de la base de datos y corroborar si hay conexión o no.

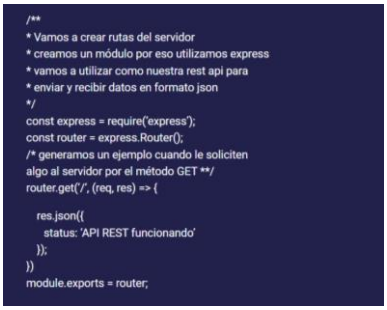
Por ser un módulo que se utiliza en toda la aplicación se exporta. De esta forma la constante Mongoose devolverá la conexión.

De manera adicional se puede observar que en el archivo index.js, se invoca o llama el archivo database.js en el cual se especifica que lo que se requiere es la conexión. En este sentido, solo resta probar si la conexión está activa, por lo que

se inicia el servidor con la línea de comando npm run dev, vista anteriormente. A continuación, se muestra en la figura el resultado de la terminal del VS Code:

1.5 Configuración de las rutas del proyecto con Node.js

El siguiente paso es configurar las rutas o URL por donde se van a enviar o recibir los datos, por lo que se crea un archivo llamado empleado.route.js dentro de la carpeta *routes* (puede descargar el contenido de estas instrucciones en el archivo empleado.route.js disponible en la carpeta de anexos). En este primer apartado lo que se quiere es revisar en primera instancia es el funcionamiento de las rutas, por lo que a manera de prueba se tiene el código que se muestra en la siguiente figura:



```
/**
* Vamos a crear rutas del servidor
* creamos un módulo por eso utilizamos express
* vamos a utilizar como nuestra rest api para
* enviar y recibir datos en formato json
*/
const express = require('express');
const router = express.Router();
/* generamos un ejemplo cuando le soliciten
algo al servidor por el método GET **/
router.get('/', (req, res) => {

  res.json({
    status: 'API REST funcionando'
  });
})
module.exports = router;
```

De manera adicional se puede observar el archivo index.js, en el cual se establece la ruta por donde se van a recibir o enviar los datos desde el servidor que para el caso de estudio será **http://localhost:3000/api/empleados**, de tal forma se observa que para la definición de esta ruta se hace necesario requerir el archivo empleado.routes.js ejecutando la siguiente instrucción:

```
app.use('/api/empleados',require('./routes/empleado.routes'));
```

Ya visto un ejemplo del funcionamiento de las rutas es necesario comprender su comportamiento, por lo que cada solicitud debe ir asociada a una ruta, y esta a su vez a un intercambio de información o de datos, es decir, de interacción con la BD. En ese sentido, es más que necesario la construcción del modelo empleado.js con datos específicos para los empleados dentro de la carpeta *models* (puede descargar el contenido de estas instrucciones en el archivo empleado.js disponible en la carpeta de anexos).

P12. *Estructura empleado.js*

Se requiere Mongoose y se almacena en una constante denominada de la misma forma, que se utilizará para definir los esquemas de datos, para lo cual se empiezan a modelar con su información, nombre, cargo, entre otros. En ese sentido, Mongoose le dice a MongoDB cómo va a lucir la estructura de empleados.

```
const mongoose = require('mongoose');
const {Schema} = mongoose;

const EmpleadoSchema = new Schema({
    name: {type:String, require:true},
    position: {type:String, require:true},
    office: {type:String, require:true},
    salary: {type:Number, require:true},
});

module.exports = mongoose.model('Empleado', EmpleadoSchema);
```

1.6 Desarrollo de controladores para la API REST

Una de las mejores prácticas para estructurar una aplicación cuando va en crecimiento es el uso de los controladores. Estos se encargan de definir los métodos y acciones que necesitan las rutas. Por lo tanto, se crea el archivo empleado.controller.js dentro de la carpeta *controllers* (puede descargar el contenido de estas instrucciones en el archivo empleado.controller.js disponible en la carpeta de anexos).

```
* Se coloca el controlador como un objeto y luego se exporta como
* se requiere primero el modelo empleado
*/

const Empleado = require('../models/empleado');
const empleadoCtrl = {};

/**
* DEFINO LOS MÉTODOS */

//Obtener todos los empleados
empleadoCtrl.getEmpleados = async (req, res) => {
    const empleados = await Empleado.find();
    res.json(empleados);
}

// Crear empleados

empleadoCtrl.createEmpleados = async (req, res) => {
    const empleado = new Empleado(req.body);
    await empleado.save();
    res.json({
      'status': 'Empleado guardado'
    });
```

Finalmente, lo que se hace al crear un módulo es exportarlo para que sea utilizado en cualquier parte de la aplicación. A continuación, se definen las rutas, por lo que se configura y desarrolla la lógica de esta sección en el archivo empleados.routes.js dentro de la carpeta *routes* (puede descargar el contenido de estas instrucciones en el archivo empleados.routes.js disponible en la carpeta de anexos).

```
/**
*
* creamos un módulo por eso utilizamos express
*/

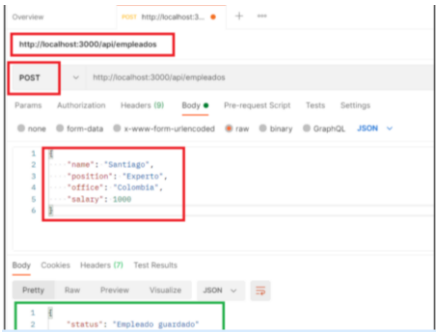
const express = require('express');
const router = express.Router();
const empleadoCtrl = require('../controllers/empleado.controller');

router.get('/', empleadoCtrl.getEmpleados); // Rutas más limpias (obtener empleados)
router.post('/', empleadoCtrl.createEmpleados);//guardar
router.get('/:id', empleadoCtrl.getUnicoEmpleado);// obtiene un único empleado
router.put('/:id',empleadoCtrl.editarEmpleado); //Actualizar datos (uno a la vez)
router.delete('/:id', empleadoCtrl.eliminarEmpleado);

module.exports = router;
```

P13. 1.7 Realización de pruebas de la API REST con Postman

Una vez finalizada la construcción tanto del modelo, controlador y rutas, se dispone a generar las pruebas respectivas para evaluar el funcionamiento de la API REST; por lo tanto, se utilizará la herramienta Postman. En la sección material complementario de este documento encontrará la guía para su instalación.



Se inicia insertando datos, por lo que utilizará el método *POST*, el cual llamará a la ruta para crear empleados.

En la figura anterior se pueden apreciar los datos que se envían al servidor, dado que Postman permite realizar esa emulación del cliente. Los datos que se envían están acordes al esquema definido en el modelo de empleado.js, como lo son *name, position, office y salary*. La respuesta en consola, que se programa como *status* es que el Empleado ha sido guardado.

En este orden de ideas, es necesario revisar la base de datos y verificar que efectivamente se ha guardado la información y si realmente MongoDB la ha creado.



Como es evidente la API REST funciona correctamente, se ha creado la base de datos empleado y se tiene almacenado el primer registro del empleado Santiago. De igual forma, se evidencia para los demás métodos *HTTP, GET, PUT o DELETE* que también funcionan, por lo cual se realiza la invitación a probar el API con Postman.

Perfecto, aquí tienes un **ejemplo completo de Frontend + Backend + Base de Datos**, fácil para practicar en **Visual Studio Code**.
Usaremos:

- **Backend**: Node.js + Express
- **Base de datos**: SQLite (muy sencilla, sin instalar servidor)
- **Frontend**: HTML + JavaScript (fetch)

📂 1. Estructura del proyecto

mi-proyecto/

│

├── backend/

│ ├── app.js

│ └── database.js

│

└── frontend/

└── index.html

🗄️ 2. Base de datos (SQLite)

📌 Usaremos SQLite porque no requiere instalación.
En el backend instalamos el módulo:

En **/backend**, abre una terminal:

npm init -y

npm install express cors sqlite3

🧠 3. Crear database.js

Archivo: **backend/database.js**

```
const sqlite3 = require("sqlite3").verbose();

// Crear base de datos (archivo local)
const db = new sqlite3.Database("./miBD.db");

// Crear tabla si no existe
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS productos (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      nombre TEXT NOT NULL,
      precio REAL NOT NULL
    );
  `);

  // Insertar datos iniciales solo si está vacío
  db.get("SELECT COUNT(*) as count FROM productos", (err, row) => {
    if (row.count === 0) {
      db.run("INSERT INTO productos (nombre, precio) VALUES ('Arroz', 2500)");
      db.run("INSERT INTO productos (nombre, precio) VALUES ('Aceite', 5800)");
    }
  });
});

module.exports = db;
```

⚙️ 4. Crear el Backend app.js

Archivo: **backend/app.js**

```
const express = require("express");
const cors = require("cors");
const db = require("./database");
```

```
const app = express();
app.use(cors());
app.use(express.json());

// Obtener productos
app.get("miprotecto/productos", (req, res) => {
  db.all("SELECT * FROM productos", [], (err, rows) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json(rows);
  });
});

// Agregar producto
app.post("miproyecto/productos", (req, res) => {
  const { nombre, precio } = req.body;

  db.run(
    "INSERT INTO productos (nombre, precio) VALUES (?, ?)",
    [nombre, precio],
    function (err) {
      if (err) return res.status(500).json({ error: err.message });

      res.json({ id: this.lastID, nombre, precio });
    }
  );
});

app.listen(3000, () => {
  console.log("Servidor corriendo en http://localhost:3000");
});
```

🎨 5. Crear el Frontend index.html

Archivo: **frontend/index.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Frontend + Backend + BD</title>
</head>
<body>
  <h1>Productos desde la Base de Datos</h1>

  <button id="btn-cargar">Cargar Productos</button>

  <ul id="lista"></ul>

  <h2>Agregar Producto</h2>
  <input id="nombre" placeholder="Nombre">
  <input id="precio" placeholder="Precio" type="number">
  <button id="btn-agregar">Agregar</button>

  <script>
    document.getElementById("btn-cargar").onclick = async () => {
      const res = await fetch("http://localhost:3000/productos");
      const productos = await res.json();

      const lista = document.getElementById("lista");
      lista.innerHTML = "";
      productos.forEach(p => {
        const li = document.createElement("li");
        li.textContent = `${p.id} - ${p.nombre} ($${p.precio})`;
```

```
            lista.appendChild(li);
        });
    };

    document.getElementById("btn-agregar").onclick = async () => {
        const nombre = document.getElementById("nombre").value;
        const precio = document.getElementById("precio").value;

        await fetch("http://localhost:3000/productos", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ nombre, precio })
        });

        alert("Producto agregado");
    };
  </script>
</body>
</html>
```

▶️ 6. Cómo ejecutarlo

1️⃣ Iniciar Backend

En la carpeta **backend**:

node app.js

Debe aparecer:

Servidor corriendo en http://localhost:3000

2️⃣ Abrir Frontend

Abre el archivo:

frontend/index.html

Doble clic o con la extensión **Live Server**.

🎉 ¿Qué puedes hacer con este ejemplo?

✓ Ver productos almacenados en la base de datos
✓ Insertar nuevos productos
✓ Visualizar los datos desde el frontend
✓ Aprender el flujo **Frontend → Backend → BD**

¡Perfecto! Vamos a mejorar el **frontend** agregando un formulario bonito y moderno usando **CSS puro**, sin frameworks.

A continuación tienes un **index.html** completo y listo para usar, con:

✓ Diseño moderno
✓ Tarjetas (cards)
✓ Botones estilizados
✓ Campos de formulario limpios
✓ Tipografía agradable
✓ Estructura organizada

🎨 Frontend con formulario bonito (HTML + CSS)

Cambia tu archivo **frontend/index.html** por este:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Productos - Frontend + Backend + BD</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f2f5;
      margin: 0;
      padding: 20px;
      display: flex;
      justify-content: center;
    }

    .container {
      width: 500px;
```

```
}

h1, h2 {
    text-align: center;
    color: #333;
}

/* Card */
.card {
    background: #fff;
    padding: 20px;
    border-radius: 12px;
    box-shadow: 0 4px 20px rgba(0,0,0,0.1);
    margin-bottom: 20px;
}

/* Botón */
.btn {
    background-color: #4CAF50;
    color: white;
    padding: 10px 16px;
    border-radius: 8px;
    border: none;
    cursor: pointer;
    width: 100%;
    font-size: 16px;
    transition: 0.3s;
}

.btn:hover {
    background-color: #45a049;
```

```
}

/* Formulario */
.form-group {
    margin-bottom: 15px;
}

label {
    font-weight: bold;
    display: block;
    margin-bottom: 6px;
}

input {
    width: 100%;
    padding: 10px;
    border-radius: 6px;
    border: 1px solid #ccc;
    font-size: 14px;
    transition: 0.2s;
}

input:focus {
    border-color: #4CAF50;
    outline: none;
    box-shadow: 0 0 6px rgba(76, 175, 80, 0.4);
}

.lista-productos {
    padding: 0;
    list-style: none;
```

```
        }

        .producto-item {
            background: #fff;
            padding: 12px;
            margin-bottom: 10px;
            border-radius: 8px;
            box-shadow: 0 3px 10px rgba(0,0,0,0.08);
            font-size: 16px;
        }

    </style>
</head>
<body>

<div class="container">

    <h1>Inventario de Productos</h1>

    <!-- Card para mostrar productos -->
    <div class="card">
        <button id="btn-cargar" class="btn">Cargar Productos</button>
        <ul id="lista" class="lista-productos"></ul>
    </div>

    <!-- Card para agregar productos -->
    <div class="card">
        <h2>Agregar Producto</h2>

        <div class="form-group">
            <label>Nombre del producto:</label>
```

```
      <input id="nombre" type="text" placeholder="Ej: Arroz">
    </div>

    <div class="form-group">
      <label>Precio:</label>
      <input id="precio" type="number" placeholder="Ej: 2500">
    </div>

    <button id="btn-agregar" class="btn">Agregar Producto</button>
  </div>
</div>

<script>
  document.getElementById("btn-cargar").onclick = async () => {
    const res = await fetch("http://localhost:3000/productos");
    const productos = await res.json();

    const lista = document.getElementById("lista");
    lista.innerHTML = "";

    productos.forEach(p => {
      const li = document.createElement("li");
      li.className = "producto-item";
      li.textContent = `${p.id} - ${p.nombre} ($${p.precio})`;
      lista.appendChild(li);
    });
  };

  document.getElementById("btn-agregar").onclick = async () => {
    const nombre = document.getElementById("nombre").value;
    const precio = document.getElementById("precio").value;
```

```
    if (!nombre || !precio) {
      alert("Por favor completa todos los campos.");
      return;
    }

    await fetch("http://localhost:3000/productos", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ nombre, precio })
    });

    alert("Producto agregado correctamente");

    document.getElementById("nombre").value = "";
    document.getElementById("precio").value = "";
  };
</script>

</body>
</html>
```

🎉 **Resultado**

Con este diseño ahora tendrás:

✨ Tarjetas con sombras
✨ Botones modernos
✨ Inputs con efecto focus verde
✨ Lista visualmente atractiva
✨ Una interfaz limpia y profesional

Perfecto para presentar en clase o usar como base en un proyecto real.