


	<i>Prueba Técnica</i>	
	Versión 1.0	

Diego David Cap Rosales - dcap@bdgsa.net

Prueba Técnica para Desarrollador Jr. (Duración: 1 hora)

	Manual de instalación	
	Versión 1.0	Página 2 de 8

Instrucciones Generales:

1. Puedes usar cualquier editor de texto o IDE.
2. Entrega el código en un archivo comprimido o súbelo a un repositorio de GitHub con instrucciones para ejecutar el proyecto.
3. Se evaluará funcionalidad, claridad del código y uso de buenas prácticas.

Parte 1: Node.js + MongoDB (30 minutos)

Problema: API básica para gestión de productos

Crea una API REST usando Node.js y Express que permita manejar un catálogo de productos. La información de los productos se almacenará en una base de datos MongoDB.

Requerimientos:

1. GET /products

Devuelve una lista de todos los productos en formato JSON.

Ejemplo de salida:

```
[
  { "id": "1", "name": "Producto A", "price": 100 },
  { "id": "2", "name": "Producto B", "price": 200 }
]
```

2. POST /products

Recibe un objeto JSON con los campos name y price, y agrega un nuevo producto a la base de datos.

Ejemplo de entrada:

```
{ "name": "Producto C", "price": 300 }
```

3. DELETE /products/:id

Elimina un producto específico por su ID.

Consideraciones:

- Usa Mongoose para conectarte a MongoDB.
- Valida que los campos name y price no estén vacíos.
- Devuelve un mensaje de error si se intenta eliminar un producto que no existe.

R//

```
dbjs > dbjs > @ connectDB
1 // config/db.js
2 const mongoose = require('mongoose');
3
4 const connectDB = async () => {
5   try {
6     // Conexion a la base de datos MongoDB
7     await mongoose.connect('mongodb://127.0.0.1:27017/catalogo');
8     console.log('MongoDB conectado exitosamente');
9   } catch (error) {
10    console.error('Error al conectar a MongoDB:', error.message);
11    process.exit(1);
12  }
13 };
14
15 module.exports = connectDB;
```

```
dbjs > server.js > ...
1 // server.js
2 const express = require('express');
3 const bodyParser = require('body-parser');
4 const cors = require('cors');
5 const connectDB = require('./config/db');
6 const productRoutes = require('./routes/productRoutes');
7
8 const app = express();
9 const PORT = 3000;
10
11 // Conectar a la base de datos
12 connectDB();
13
14
15 app.use(bodyParser.json());
16 app.use(cors());
17
18 // Rutas
19 app.use('/api', productRoutes);
20
21 // Servidor
22 app.listen(PORT, () => {
23   console.log('Servidor corriendo en http://localhost:${PORT}');
24 });
25
```

```
dbjs > productRoutes.js > @ router.post('/products') callback > @ newProduct
1 // routes/productRoutes.js
2 const express = require('express');
3 const router = express.Router();
4 const Product = require('../models/Product');
5
6 // GET: Devuelve todos los productos
7 router.get('/products', async (req, res) => {
8   try {
9     const products = await Product.find();
10    res.json(products);
11  } catch (error) {
12    res.status(500).json({ message: 'Error al obtener los productos' });
13  }
14 });
15
16 // POST: Agrega un nuevo producto
17 router.post('/products', async (req, res) => {
18   try {
19     const { name, price } = req.body;
20     if (!name || !price) {
21       return res.status(400).json({ message: 'Name y price son requeridos' });
22     }
23
24     const newProduct = new Product({ name, price });
25     await newProduct.save();
26     res.status(201).json(newProduct);
27   } catch (error) {
28     res.status(500).json({ message: 'Error al agregar el producto' });
29   }
30 });
31
32 // DELETE: Elimina un producto por ID
33 router.delete('/products/:id', async (req, res) => {
34   try {
35     const product = await Product.findByIdAndDelete(req.params.id);
36     if (!product) {
37       return res.status(404).json({ message: 'Producto no encontrado' });
38     }
39     res.json({ message: 'Producto eliminado correctamente' });
40   } catch (error) {
41     res.status(500).json({ message: 'Error al eliminar el producto' });
42   }
43 });
44
45 module.exports = router;
```

```
db.js  server.js  productRoutes.js  Product.js X
models > Product.js > ...
1
2 const mongoose = require('mongoose');
3
4 const ProductSchema = new mongoose.Schema({
5   name: { type: String, required: true },
6   price: { type: Number, required: true }
7 });
8
9 module.exports = mongoose.model('Product', ProductSchema);
10
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
(node:5880) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Servidor corriendo en http://localhost:3000
Error al conectar a MongoDB connect ECONNREFUSED 127.0.0.1:27017
P5 C:\Users\lenov\Downloads\api-productos> node server.js
Servidor corriendo en http://localhost:3000
```

```
app.component.html M X
frontend-productos > src > app > app.component.html > div.container > table
336 <!--<router-outlet /> -->
337
338 <div class="container">
339   <h1>Catálogo de Productos</h1>
340   <table>
341     <thead>
342       <tr>
343         <th>Nombre</th>
344         <th>Precio</th>
345         <th>Acciones</th>
346       </tr>
347     </thead>
348     <tbody>
349       <tr *ngFor="let product of products">
350         <td>{{ product.name }}</td>
351         <td>{{ product.price }}</td>
352         <td>
353           <button (click)="deleteProduct(product._id)">Eliminar</button>
354         </td>
355       </tr>
356     </tbody>
357   </table>
358 </div>
359 <h1>Hello, {{ title }}</h1>
360 <router-outlet></router-outlet>
361
```

Parte 2: Angular (20 minutos)

Problema: Interfaz básica para mostrar productos

Crea una aplicación Angular que consuma la API creada en la Parte 1 y muestre los productos en una tabla.

Requerimientos:

1. Usa Angular CLI para generar un nuevo proyecto.
2. Implementa un servicio que consuma la ruta GET /products de la API.
3. Muestra los productos en una tabla con las columnas Nombre y Precio.

4. Agrega un botón para eliminar un producto desde la tabla. Cuando se presione, debe enviar una solicitud DELETE a la API y actualizar la tabla.

Consideraciones:

- Usa Angular Material para diseñar la tabla (opcional).
- Asegúrate de manejar errores básicos (por ejemplo, mostrar un mensaje si la API no responde).

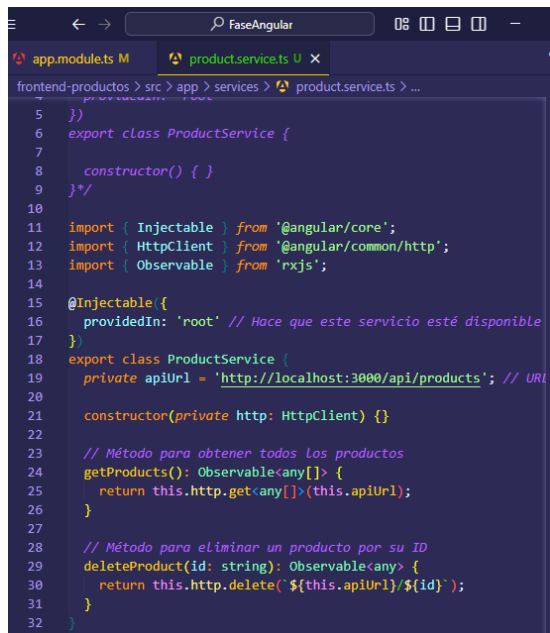
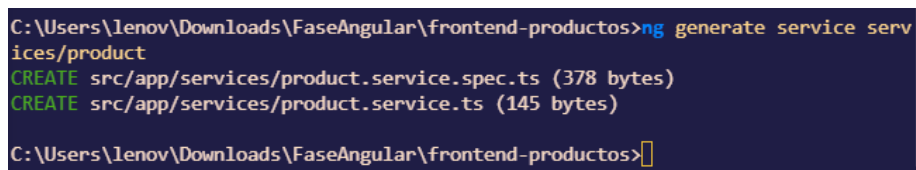
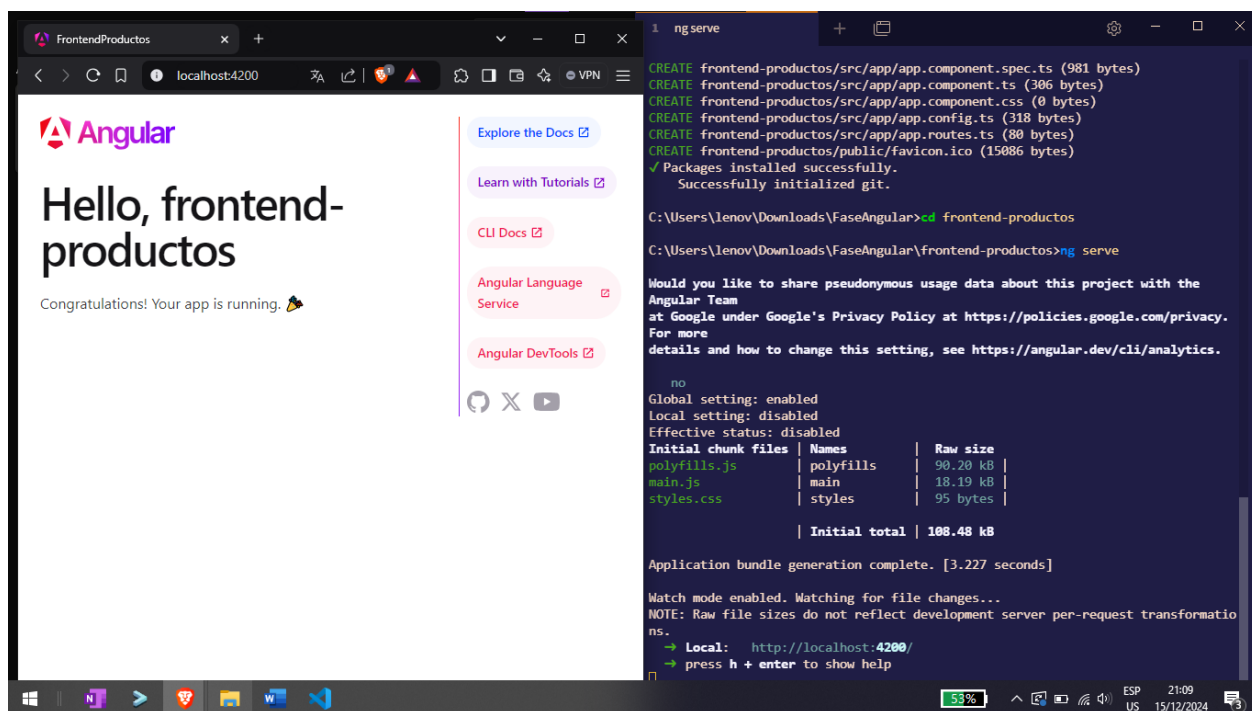
R//

```
C:\Users\lenov>ng version

Angular CLI
Angular CLI: 19.0.5
Node: 22.12.0
Package Manager: npm 9.7.2
OS: win32 x64

Angular:
...
Package      Version
-----
@angular-devkit/architect    0.1900.5 (cli-only)
@angular-devkit/core         19.0.5 (cli-only)
@angular-devkit/schematics   19.0.5 (cli-only)
@schematics/angular          19.0.5 (cli-only)

C:\Users\lenov>
```



```
product.service.ts U  app.component.ts M X
frontend-productos > src > app > app.component.ts > ...
14 import { Component, OnInit } from '@angular/core';
15 import { ProductService } from '../services/product.service';
16
17 @Component({
18   selector: 'app-root',
19   templateUrl: './app.component.html',
20   styleUrls: ['./app.component.css']
21 })
22 export class AppComponent implements OnInit {
23   products: any[] = []; // Lista de productos
24
25   constructor(private productService: ProductService) {}
26
27   ngOnInit(): void {
28     this.fetchProducts();
29   }
30
31   // Obtener Los productos desde La API
32   fetchProducts(): void {
33     this.productService.getProducts().subscribe(
34       (data) => {
35         this.products = data; // Guardar Los productos en la variable
36       },
37       (error) => {
38         console.error('Error al obtener productos:', error);
39       }
40     );
41   }
42
43   // Eliminar un producto y actualizar la tabla
44   deleteProduct(id: string): void {
45     this.productService.deleteProduct(id).subscribe(
46       () => {
47         this.fetchProducts(); // Actualiza la lista despues de eliminar
48       },
49       (error) => {
50         console.error('Error al eliminar producto:', error);
51       }
52     );
53   }
54 }
```

Parte 3: SQL (10 minutos)

Problema: Consultas SQL básicas

Dada la siguiente tabla llamada: **orders**:


id	product_name	quantity	total_price
1	Producto A	2	200
2	Producto B	1	200
3	Producto A	3	300

Responde las siguientes preguntas escribiendo las consultas SQL correspondientes:

1. Obtener todas las órdenes donde la cantidad sea mayor a 1.

R// SELECT * FROM orders WHERE quantity > 1;

2. Calcular el total de ingresos (SUM (total_price)).

	Manual de instalación	
	Versión 1.0	Página 8 de 8

R// SELECT SUM(total_price) AS total_ingresos FROM orders;

3. Contar cuántas órdenes contienen el producto "Producto A".

R// SELECT COUNT(*) AS cantidad_ordenes FROM orders WHERE product_name = 'Producto A';

Entrega

- **Node.js y MongoDB:** Subir el código a un repositorio con un archivo README para explicar cómo instalar y ejecutar.
- **Angular:** Incluye el proyecto Angular en el mismo repositorio o en uno separado.
- **SQL:** Adjuntar un archivo de texto con las consultas SQL.

Evaluación Global:

- Completitud de las funcionalidades.
- Uso adecuado de las tecnologías (servicios en Angular, rutas en Node.js).
- Claridad y organización del código.