



Solución: Taller Práctico de Integración FullStack

Julian F. Latorre
Bootcamp Desarrollo Web Full Stack

15 de noviembre de 2024

Sobre la Solución

Este documento presenta la solución completa al taller de integración FullStack. Se incluye el código comentado, explicaciones detalladas y mejores prácticas implementadas en cada sección.

1. Estructura del Proyecto Implementada

Estructura Final

```
mi-proyecto/  
├── frontend/  
│   ├── index.html  
│   ├── styles.css  
│   ├── app.js  
│   └── assets/  
│       ├── img/  
│       └── fonts/  
└── backend/  
    ├── server.js  
    ├── package.json  
    ├── routes/  
    │   └── notes.js  
    ├── middleware/  
    │   ├── auth.js  
    │   └── validator.js  
    └── config/  
        ├── database.js  
        └── .env
```

2. Frontend Implementado

2.1. HTML Base

Archivo: index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
  <title>Mi App de Notas</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Mis Notas</h1>
    <div id="notas-form" class="form-container">
      <input type="text" id="nota-titulo"
        placeholder="Título" required>
      <textarea id="nota-contenido"
        placeholder="Contenido" required></
    textarea>
      <button onclick="crearNota()"
        class="btn-primary">Crear Nota</button>
    </div>
    <div id="notas-lista" class="notes-grid"></div>
  </div>
  <script src="app.js"></script>
</body>
</html>
```

2.2. Estilos CSS

Archivo: styles.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: Arial, sans-serif;
  line-height: 1.6;
  background-color: #f5f5f5;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

.form-container {
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  margin-bottom: 20px;
}

input, textarea {
  width: 100%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

textarea {
  height: 100px;
  resize: vertical;
}
```

Archivo: styles.css

```
.btn-primary {
  background-color: #007bff;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.notes-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 20px;
}

.nota {
  background: white;
  padding: 15px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

@media (max-width: 600px) {
  .container {
    padding: 10px;
  }

  .notes-grid {
    grid-template-columns: 1fr;
  }
}
```

2.3. JavaScript Frontend

Archivo: app.js

```
const API_URL = 'http://localhost:3000';
let notes = [];

// Inicialización
document.addEventListener('DOMContentLoaded', () => {
  cargarNotas();
  inicializarValidacion();
});

function inicializarValidacion() {
  const titulo = document.getElementById('nota-titulo');
  const contenido = document.getElementById('nota-contenido');

  titulo.addEventListener('input', validarCampo);
  contenido.addEventListener('input', validarCampo);
}

async function crearNota() {
  const titulo = document.getElementById('nota-titulo');
  const contenido = document.getElementById('nota-contenido');

  if (!validarFormulario(titulo, contenido)) return;

  try {
    const response = await fetch(`${API_URL}/notas`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        titulo: titulo.value,
        contenido: contenido.value
      })
    });

    if (!response.ok) throw new Error('Error al crear nota');

    const nota = await response.json();
    guardarEnLocalStorage(nota);
    await cargarNotas();
    limpiarFormulario();
  } catch (error) {
    mostrarError('Error al crear la nota');
  }
}
```

Archivo: app.js

```
async function cargarNotas() {
  try {
    const notasLocal = obtenerDeLocalStorage();
    if (notasLocal) renderizarNotas(notasLocal);

    const response = await fetch(`${API_URL}/notas`);
    if (!response.ok) throw new Error('Error al cargar
notas');

    const notas = await response.json();
    renderizarNotas(notas);
    actualizarLocalStorage(notas);
  } catch (error) {
    mostrarError('Error al cargar las notas');
  }
}

// Funciones auxiliares implementadas...
```


3. Backend Implementado

3.1. Servidor Express

Archivo: server.js

```
const express = require('express');
const cors = require('cors');
const { v4: uuidv4 } = require('uuid');
const { validarNota } = require('../middleware/validator');
const app = express();

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

let notas = [];

app.post('/notas', validarNota, (req, res) => {
  try {
    const nuevaNota = {
      id: uuidv4(),
      ...req.body,
      fecha: new Date()
    };

    notas.push(nuevaNota);
    res.status(201).json(nuevaNota);
  } catch (error) {
    res.status(500).json({
      error: 'Error al crear la nota'
    });
  }
});

app.get('/notas', (req, res) => {
  res.json(notas);
});

app.delete('/notas/:id', (req, res) => {
  try {
    const { id } = req.params;
    const index = notas.findIndex(nota => nota.id === id);
    ;

    if (index === -1) {
      return res.status(404).json({
        error: 'Nota no encontrada'
      });
    }

    notas.splice(index, 1);
    res.status(200).json({
      mensaje: 'Nota eliminada correctamente'
    });
  } catch (error) {
    res.status(500).json({
      error: 'Error al eliminar la nota'
    });
  }
});

app.listen(3000, () => {
  console.log('Servidor corriendo en puerto 3000');
});
```


3.2. Validador

Archivo: middleware/validator.js

```
function validarNota(req, res, next) {
  const { titulo, contenido } = req.body;

  if (!titulo || !contenido) {
    return res.status(400).json({
      error: 'El título y contenido son requeridos'
    });
  }

  if (titulo.length > 100) {
    return res.status(400).json({
      error: 'El título no puede exceder 100 caracteres'
    });
  }

  if (contenido.length > 1000) {
    return res.status(400).json({
      error: 'El contenido no puede exceder 1000 caracteres'
    });
  }

  next();
}

module.exports = { validarNota };
```

4. Retos Implementados

4.1. Eliminación de Notas

Se implementó la funcionalidad de eliminación mediante:

- Endpoint DELETE en el backend
- Botón de eliminación en cada nota
- Actualización del localStorage
- Manejo de errores

4.2. Validación de Datos

Se implementó validación en:

- Frontend: Validación en tiempo real
- Backend: Middleware de validación
- Límites de caracteres
- Campos requeridos

4.3. Persistencia LocalStorage

Se implementó persistencia mediante:

- Guardado automático en localStorage
- Sincronización con backend
- Fallback cuando no hay conexión
- Actualización en tiempo real

4.4. Diseño Responsivo

Se implementó diseño responsivo mediante:

- CSS Grid con auto-fill
- Media queries
- Flexbox para layouts
- Unidades relativas

5. Respuestas a Preguntas Guía

¿Cómo manejas los errores en las peticiones fetch?

- Try/catch en todas las operaciones asíncronas
- Verificación de response.ok
- Mensajes de error específicos
- Fallback a datos locales

¿Qué medidas de seguridad implementaste?

- Validación de entrada en frontend y backend
- Sanitización de datos
- CORS configurado
- Límites en tamaños de datos

¿Cómo mejoraste el rendimiento?

- Caché local con localStorage
- Optimización de renders
- Lazy loading de datos
- CSS eficiente con Grid

6. Mejores Prácticas Implementadas

6.1. Frontend

- Separación de responsabilidades en módulos
- Manejo de estado consistente
- Validación en tiempo real
- Feedback inmediato al usuario
- Diseño responsivo y accesible
- Manejo de errores amigable

6.2. Backend

- Arquitectura modular
- Middleware para validación
- Manejo de errores centralizado
- Respuestas HTTP apropiadas
- CORS configurado correctamente
- Logging de errores

7. 🐙 Guía de Despliegue

7.1. Frontend

Pasos para Despliegue Frontend

1. Verificar configuración de API_URL para producción:

```
const API_URL = process.env.API_URL || 'http://localhost:3000';
```

2. Minificar archivos estáticos:

```
# Usando herramientas como webpack o parcel  
npm run build
```

3. Configurar servidor web (ejemplo con nginx):

```
server {  
    listen 80;  
    root /var/www/notas-app;  
    index index.html;  
  
    location / {  
        try_files $uri $uri/ /index.html;  
    }  
}
```

7.2. Backend

Pasos para Despliegue Backend

1. Configurar variables de entorno:

```
# .env
PORT=3000
NODE_ENV=production
```

2. Preparar script de inicio:

```
// package.json
{
  "scripts": {
    "start": "node server.js",
    "prod": "NODE_ENV=production node server.js"
  }
}
```

3. Configurar PM2 para gestión de procesos:

```
pm2 start server.js --name "notas-api"
pm2 save
```

8. Pruebas Realizadas

8.1. Pruebas Funcionales

- Creación exitosa de notas
- Visualización correcta en grid
- Eliminación de notas
- Persistencia en localStorage
- Sincronización backend-frontend

8.2. Pruebas de Validación

- Campos vacíos
- Límites de caracteres
- Tipos de datos incorrectos
- Caracteres especiales

8.3. Pruebas de Responsividad

- Dispositivos móviles
- Tablets
- Escritorio
- Diferentes navegadores

9. Conclusiones y Siguietes Pasos

9.1. Logros Alcanzados

- Implementación completa de CRUD
- Interfaz responsiva y amigable
- Sistema robusto de manejo de errores
- Persistencia de datos efectiva

9.2. Mejoras Futuras

- Implementar autenticación de usuarios
- Agregar categorías para notas
- Implementar búsqueda y filtros
- Añadir soporte para markdown
- Implementar sincronización en tiempo real

10. Referencias y Recursos

- MDN Web Docs - fetch API
- Express.js Documentation
- CSS Grid Layout Guide
- localStorage Web API
- CORS Configuration Guide