# Guía Completa de Implementación - Sistema de Gestión de Portafolios de Cursos

**Documentación Técnica**  
**Versión:** 1.0  
**Fecha:** [Fecha Actual]

## 1. Introducción

Este documento proporciona una guía paso a paso para implementar el sistema de gestión de portafolios de cursos utilizando tecnologías web (HTML, CSS, JavaScript) y SQLite como base de datos. El sistema está diseñado para funcionar en un entorno local con autenticación de usuarios, gestión de cursos, portafolios y archivos.

**Explicación:** Esta introducción establece el propósito del sistema. Se trata de una aplicación web que permitirá gestionar portafolios académicos con diferentes roles de usuario (administradores, docentes, evaluadores). La elección de SQLite es perfecta para desarrollo local ya que es liviana y no requiere configuración de servidor.

## 2. Requisitos Previos

Antes de comenzar, asegúrate de tener instalado:

* **Node.js (v16 o superior)** [Descargar aquí](https://nodejs.org/)
* **NPM** (viene con Node.js)
* **Visual Studio Code** (u otro editor de código) [Descargar aquí](https://code.visualstudio.com/)
* **Extensión "Live Server"** para VSCode (opcional, para previsualización)

**Explicación de cada requisito:**

* **Node.js**: Es el entorno de ejecución que nos permitirá usar JavaScript en el backend y manejar la base de datos SQLite
* **NPM**: El gestor de paquetes que nos ayudará a instalar las dependencias necesarias (sqlite3, bcryptjs)
* **Visual Studio Code**: Editor recomendado por su integración con extensiones web
* **Live Server**: Extensión que crea un servidor local para visualizar la aplicación en tiempo real

## 3. Estructura del Proyecto

El proyecto tiene la siguiente estructura:

/proyecto-portafolio/

│

├── portafolio.db # Base de datos SQLite

├── database.js # Configuración de la base de datos

│

├── index.html # Página principal

├── login.html # Página de inicio de sesión

├── dashboard.html # Panel de control según rol

│

├── css/

│ └── style.css # Estilos principales

│

├── js/

│ ├── auth.js # Autenticación de usuarios

│ ├── cursos.js # Gestión de cursos

│ ├── portafolio.js # Gestión de portafolios

│ └── archivos.js # Subida/descarga de archivos

│

└── uploads/ # Carpeta para archivos subidos

**Explicación de la estructura:**

* **Raíz del proyecto**: Contiene archivos principales y la base de datos
* **css/**: Carpeta dedicada a los estilos visuales
* **js/**: Scripts organizados por funcionalidad específica
* **uploads/**: Almacenamiento físico de los archivos que suban los usuarios

## 4. Pasos para Implementación

### 4.1. Configuración Inicial

**Paso 1: Crear la carpeta del proyecto**

mkdir proyecto-portafolio

cd proyecto-portafolio

**Explicación:** Creamos el directorio raíz y nos posicionamos en él para trabajar.

**Paso 2: Inicializar Node.js**

npm init -y

**Explicación:** Este comando crea el archivo package.json que contendrá la información del proyecto y sus dependencias. La bandera -y acepta todas las opciones por defecto.

**Paso 3: Instalar dependencias**

npm install sqlite3 bcryptjs

**Explicación:**

* sqlite3: Biblioteca para manejar la base de datos SQLite desde Node.js
* bcryptjs: Para encriptar contraseñas de forma segura

**Paso 4: Crear la estructura de carpetas**

mkdir css js uploads

**Explicación:** Creamos las carpetas necesarias para organizar el código y archivos.

### 4.2. Configuración de la Base de Datos

**Crear el archivo database.js** (basado en tu init.js original):

const sqlite3 = require('sqlite3').verbose();

const path = require('path');

const bcrypt = require('bcryptjs');

const dbPath = path.join(\_\_dirname, 'portafolio.db');

const db = new sqlite3.Database(dbPath);

// Función para inicializar tablas

const initDB = () => {

db.serialize(() => {

// Crear tablas (usuarios, cursos, portafolios, etc.)

// ... (código completo proporcionado anteriormente)

});

};

module.exports = { db, initDB };

**Explicación del código:**

* Importamos las librerías necesarias
* Definimos la ruta donde se guardará la base de datos
* db.serialize() asegura que las operaciones se ejecuten en orden
* Exportamos la conexión y función de inicialización para usar en otros archivos

**Ejecutar la inicialización:**

**Paso 1: Crea un archivo setup.js temporal**

const { initDB } = require('./database.js');

initDB();

**Paso 2: Ejecútalo**

node setup.js

**Explicación:** Este proceso crea el archivo portafolio.db con todas las tablas necesarias (usuarios, cursos, portafolios, etc.) ya configuradas.

### 4.3. Implementación del Frontend

**Archivos HTML:**

* Copia los archivos index.html, login.html, y dashboard.html proporcionados en la raíz del proyecto
* Asegúrate de que las rutas a CSS y JS sean correctas:

<link rel="stylesheet" href="css/style.css">

<script src="js/database.js"></script>

**Explicación:** Los archivos HTML definen la estructura visual. Es crucial que las rutas sean relativas y correctas para que el navegador encuentre los recursos.

**Archivos JavaScript:**

* Coloca auth.js, cursos.js, portafolio.js y archivos.js en la carpeta js/

**Explicación:** Cada archivo maneja una funcionalidad específica, manteniendo el código organizado y modular.

**Estilos CSS:**

* Copia style.css en css/

**Explicación:** Los estilos dan la apariencia visual al sistema y aseguran una buena experiencia de usuario.

### 4.4. Conexión entre Frontend y Base de Datos

**Ejemplo de autenticación (auth.js):**

const { db } = require('./database.js');

async function loginUser(email, password) {

return new Promise((resolve, reject) => {

db.get(

"SELECT \* FROM users WHERE email = ?",

[email],

(err, user) => {

if (err) reject(err);

if (!user) reject(new Error("Usuario no encontrado"));

// ... (verificar contraseña con bcrypt)

}

);

});

}

**Explicación:** Este código muestra cómo conectar el frontend con la base de datos. Usa consultas preparadas (con ?) para prevenir inyección SQL.

**Manejo de archivos (archivos.js):**

* Los archivos se guardan en uploads/ con sus metadatos en SQLite

**Explicación:** Los archivos físicos se almacenan en el sistema de archivos, mientras que la información sobre ellos (nombre, tipo, fecha) se guarda en la base de datos.

### 4.5. Ejecución del Proyecto

**Opción 1: Usar Live Server (VSCode)**

1. Abre el proyecto en VSCode
2. Haz clic derecho en index.html > "Open with Live Server"

**Explicación:** Live Server crea un servidor local automáticamente y recarga la página cuando detecta cambios.

**Opción 2: Usar http-server**

npm install -g http-server

http-server

Accede a http://localhost:8080

**Explicación:** Alternativa que instala un servidor HTTP simple globalmente en tu sistema.

## 5. Flujo de Trabajo

### 5.1. Para Administradores

**Iniciar sesión:**

* Usuario: admin@universidad.edu
* Contraseña: admin123

**Gestionar cursos:**

* Crear/editar/eliminar cursos
* Asignar docentes

**Ver portafolios:**

* Revisar el estado de los portafolios

**Explicación:** Los administradores tienen control total del sistema y pueden gestionar todos los aspectos académicos.

### 5.2. Para Docentes

**Subir archivos:**

* Límite de 5MB por archivo
* Categorizar como "sílabo", "material", etc.

**Gestionar portafolios:**

* Cambiar estado (borrador → completado)

**Explicación:** Los docentes pueden crear y gestionar sus portafolios, subiendo materiales educativos organizados por categorías.

### 5.3. Para Evaluadores

**Revisar portafolios:**

* Ver portafolios en estado "completado"
* Dejar comentarios y calificaciones

**Explicación:** Los evaluadores pueden revisar y calificar los portafolios que los docentes han marcado como completados.

## 6. Solución de Problemas Comunes

| **Error** | **Solución** |
| --- | --- |
| "No se encuentra la tabla" | Ejecuta initDB() en database.js |
| "Error de conexión a SQLite" | Verifica que portafolio.db tenga permisos |
| "Archivo no sube" | Revisa que la carpeta uploads/ exista |

**Explicación:** Esta tabla resume los errores más frecuentes y sus soluciones rápidas durante la implementación.

## 7. Conclusiones

* El sistema está listo para funcionar localmente con SQLite
* Para producción, considera migrar a MySQL/PostgreSQL y usar un backend como Express.js
* Personaliza los estilos en style.css según necesidades

**¡Implementación completada! 🎉**

**Explicación final:** El sistema está diseñado para desarrollo y pruebas locales. Para un entorno de producción real, se recomienda usar bases de datos más robustas y frameworks de backend más completos.