PROYECTO DE VALIDACIÓN DE FORMULARIOS: HTML, CSS Y JAVASCRIPT

Vamos a crear un proyecto completo de validación de formularios que ayudará a los alumnos a entender los diferentes tipos de validaciones, desde el frontend hasta el backend.

ESTRUCTURA DEL PROYECTO

Crearemos los siguientes ejercicios progresivos:

1. Validación básica en el frontend
2. Validaciones específicas y feedback en pantalla
3. Introducción al backend con Node.js y Express
4. Validación de datos en el backend
5. Validación mixta (frontend + backend)
6. Gestión de sesiones
7. Integración con bases de datos (MongoDB)

Vamos a comenzar configurando el entorno y creando los primeros ejercicios.

CONFIGURACIÓN INICIAL

Primero, vamos a crear la estructura de carpetas para nuestro proyecto:

```
mkdir validacion-formularios
cd validacion-formularios
mkdir ejercicio1-validacion-basica
mkdir ejercicio2-validacion-especifica
mkdir ejercicio3-intro-backend
mkdir ejercicio4-validacion-backend
mkdir ejercicio5-validacion-mixta
mkdir ejercicio6-sesiones
mkdir ejercicio7-mongodb
```

EJERCICIO 1: VALIDACIÓN BÁSICA EN EL FRONTEND

Vamos a crear un formulario simple con validaciones básicas usando HTML5 y JavaScript.

1. Crea el archivo ejercicio1-validacion-basica/index.html
2. Ahora, crea el archivo ejercicio1-validacion-basica/styles.css
3. Finalmente, crea el archivo ejercicio1-validacion-basica/script.js

EJERCICIO 2: VALIDACIONES ESPECÍFICAS Y FEEDBACK EN PANTALLA

Vamos a crear un formulario más complejo con validaciones específicas y feedback visual mejorado.

1. Crea el archivo ejercicio2-validacion-especifica/index.html
2. Ahora, crea el archivo ejercicio2-validacion-especifica/styles.css
3. Finalmente, crea el archivo ejercicio2-validacion-especifica/script.js

EJERCICIO 3: INTRODUCCIÓN AL BACKEND CON NODE.JS Y EXPRESS

En este ejercicio configuraremos un entorno básico para una aplicación web sencilla que combine **frontend** (HTML, CSS, JavaScript) y **backend** (Node.js con Express).

PASO 1: INICIALIZAR PROYECTO NODE.JS

Comandos ejecutados:

```
cd ejercicio3-intro-backend
npm init -y
npm install express body-parser cors
```

Explicación detallada:

- **cd ejercicio3-intro-backend**
  Cambia el directorio actual al directorio del proyecto. Esto garantiza que todas las instalaciones y configuraciones se realicen dentro del mismo lugar, manteniendo el orden y la estructura del proyecto.

- **npm init -y**
  Inicializa un nuevo proyecto Node.js. Genera automáticamente el archivo package.json, que guarda información sobre el proyecto y las dependencias necesarias para ejecutarlo. La opción -y indica que aceptamos los valores por defecto (nombre del proyecto, versión, descripción, etc.), simplificando el proceso.

- **npm install express body-parser cors**
  Instala las dependencias esenciales:

  - **Express:** Framework web ligero y flexible para Node.js, permite crear servidores web y definir rutas fácilmente.
  - **body-parser:** Middleware que analiza y convierte las peticiones HTTP entrantes en formato JSON o formularios a objetos accesibles en JavaScript.
  - **cors:** Middleware que permite solicitudes HTTP desde dominios distintos al del servidor, resolviendo problemas de políticas de seguridad en navegadores.

PASO 2: CREAR ARCHIVO PRINCIPAL DEL SERVIDOR (SERVER.JS)

Archivo a crear:

```
ejercicio3-intro-backend/server.js
```

Explicación detallada:

- Este archivo será el **punto de entrada** de nuestra aplicación backend.
- Aquí configuraremos el servidor Node.js con Express para:
  - Definir rutas HTTP.
  - Indicar qué carpeta se usará para los archivos públicos (frontend).
  - Configurar middlewares como body-parser y cors para manejar adecuadamente las peticiones entrantes.

Contenido habitual (ejemplo):

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();

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

// Indicar la carpeta que sirve archivos estáticos (HTML, CSS, JS)
app.use(express.static('public'));

// Iniciar servidor en puerto específico
app.listen(3000, () => {
    console.log('Servidor iniciado en el puerto 3000');
});
```

PASO 3: CREAR CARPETA PÚBLICA PARA FRONTEND

Comando ejecutado:

```
mkdir ejercicio3-intro-backend/public
```

Explicación detallada:

- La carpeta public almacenará los archivos estáticos del proyecto (HTML, CSS, JS, imágenes).
- Al indicar que la carpeta public contiene archivos estáticos en Express (express.static('public')), permitimos al servidor Node.js servir automáticamente estos archivos al navegador del usuario.

PASO 4: CREAR ARCHIVO HTML PRINCIPAL (INDEX.HTML)

Archivo a crear:

```
ejercicio3-intro-backend/public/index.html
```

Explicación detallada:

- El archivo index.html es el punto inicial de carga en el navegador web.
- Contendrá la estructura HTML básica que permitirá interactuar visualmente con la aplicación web, cargar estilos CSS, y scripts JavaScript desde el frontend.

Ejemplo básico:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio 3 Backend con Node.js</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Introducción al Backend con Node.js y Express</h1>
```

```
    <script src="script.js"></script>
</body>
</html>
```

PASO 5: CREAR ARCHIVO CSS (STYLES.CSS)

Archivo a crear:

```
ejercicio3-intro-backend/public/styles.css
```

Explicación detallada:

- Este archivo separa claramente los estilos visuales de la estructura HTML.
- Proporciona flexibilidad para modificar la apariencia de la web sin afectar su estructura o funcionalidad.

Ejemplo sencillo:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f4f8;
    color: #333;
    margin: 20px;
}

h1 {
    text-align: center;
}
```

PASO 6: CREAR ARCHIVO JAVASCRIPT DEL FRONTEND (SCRIPT.JS)

Archivo a crear:

```
ejercicio3-intro-backend/public/script.js
```

Explicación detallada:

- Este archivo gestionará las interacciones dinámicas del lado del cliente (frontend).
- Permitirá realizar peticiones HTTP hacia el backend (por ejemplo, peticiones AJAX con fetch() para cargar datos dinámicamente).
- Conecta visualmente el frontend con la lógica backend definida en server.js.

Ejemplo básico:

```
document.addEventListener('DOMContentLoaded', () => {
    console.log('Frontend cargado correctamente');
});
```

Paso	Propósito fundamental
1	Prepara el proyecto Node.js e instala librerías necesarias.

2	Define la configuración inicial y lógica básica del servidor.
3	Crea estructura ordenada separando frontend (cliente) y backend.
4	Establece la estructura HTML que visualizará el usuario final.
5	Aísla estilos visuales para mantenimiento eficiente.
6	Conecta interacción dinámica entre frontend y backend.

CONCLUSIÓN

Este ejercicio te permite comprender claramente cómo se relacionan y complementan los distintos componentes de una aplicación web:

- **Node.js y Express** forman un servidor rápido y eficiente para gestionar peticiones backend.
- **HTML, CSS y JavaScript (Frontend)** definen la presentación y dinámica de usuario.
- Los archivos estructurados permiten un desarrollo limpio y fácil mantenimiento.

EJERCICIO 4: VALIDACIÓN DE DATOS EN EL BACKEND

Este ejercicio se centra en **validar datos en el backend**, esencial para asegurar que la información recibida por el servidor sea segura, correcta y fiable antes de guardarse o procesarse.

PASO INICIAL: PREPARACIÓN DEL ENTORNO DEL PROYECTO NODE.JS

Comandos ejecutados:

```
cd ejercicio4-validacion-backend
npm init -y
npm install express body-parser cors joi
```

Explicación detallada:

- **cd ejercicio4-validacion-backend**
  Establece claramente el directorio donde vamos a trabajar. Permite mantener el proyecto ordenado y claramente diferenciado de otros ejercicios.

- **npm init -y**
  Inicializa un proyecto de Node.js automáticamente generando un archivo package.json con la configuración predeterminada. Esto permite llevar un control de dependencias, scripts y metadata del proyecto.

- **npm install express body-parser cors joi**
  Instalación de dependencias clave:

  - **Express**: Framework ligero que permite crear servidores HTTP fácilmente.
  - **body-parser**: Middleware para interpretar peticiones JSON o formularios y convertirlas en objetos JavaScript accesibles.
  - **cors**: Middleware para permitir peticiones entre diferentes dominios (Cross-Origin Resource Sharing), facilitando la interacción frontend-backend.
  - **Joi**: Librería especializada en validación de datos en Node.js. Permite definir claramente esquemas de validación, asegurando la integridad de la información que llega al servidor.

PASO 1: CREAR ARCHIVO PRINCIPAL DEL SERVIDOR (SERVER.JS)

Archivo a crear:

```
ejercicio4-validacion-backend/server.js
```

Explicación detallada:

- Este archivo es el corazón del backend. Aquí configuramos nuestro servidor Express.
- Integraremos **Joi** para validar los datos recibidos desde el frontend.
- Definiremos rutas para manejar peticiones HTTP, recibir datos, validarlos con Joi y responder al frontend.

Ejemplo del contenido habitual del archivo (server.js):

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const Joi = require('joi');

const app = express();
app.use(bodyParser.json());
app.use(cors());
app.use(express.static('public'));

// Definición de una ruta para validar datos enviados desde
frontend
app.post('/datos', (req, res) => {
    // Definición del esquema de validación con Joi
    const schema = Joi.object({
        nombre: Joi.string().min(3).required(),
        edad: Joi.number().integer().min(18).max(99).required(),
        email: Joi.string().email().required()
    });

    // Validación de los datos recibidos
    const resultado = schema.validate(req.body);

    if(resultado.error) {
        return
res.status(400).send(resultado.error.details[0].message);
    }

    res.status(200).send("Datos validados correctamente.");
});

// Arrancar servidor
app.listen(3000, () => {
    console.log('Servidor ejecutándose en puerto 3000');
});
```

PASO 2: CREAR CARPETA PÚBLICA (PUBLIC) PARA EL FRONTEND

Comando ejecutado:

```
mkdir ejercicio4-validacion-backend/public
```

Explicación detallada:

- La carpeta public almacena todos los archivos que el navegador utilizará directamente (HTML, CSS, JavaScript).
- Permite una clara separación de responsabilidades:
    - **Backend:** Lógica del servidor, validaciones y gestión de datos.
    - **Frontend:** Visualización y experiencia del usuario.

PASO 3: CREAR ARCHIVO HTML PRINCIPAL (INDEX.HTML)

Archivo a crear:

```
ejercicio4-validacion-backend/public/index.html
```

Explicación detallada:

- Punto inicial visual del usuario al abrir la aplicación web.
- Contendrá un formulario HTML que enviará datos al servidor (mediante JavaScript frontend).

Ejemplo básico:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Validación Backend con Node.js</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <form id="formulario">
        <input type="text" id="nombre" placeholder="Nombre"
required>
        <input type="number" id="edad" placeholder="Edad" required>
        <input type="email" id="email" placeholder="Email"
required>
        <button type="submit">Enviar</button>
    </form>

    <div id="respuesta"></div>

    <script src="script.js"></script>
</body>
</html>
```

PASO 4: CREAR ARCHIVO CSS (STYLES.CSS)

Archivo a crear:

```
ejercicio4-validacion-backend/public/styles.css
```

Explicación detallada:

- Archivo separado que controla el diseño y apariencia visual del formulario y los elementos HTML.
- Facilita mantenimiento y organización visual.

Ejemplo básico:

```
body {
    font-family: Arial, sans-serif;
    padding: 20px;
    background-color: #f5f7fa;
```

```
}

input, button {
    display: block;
    margin: 10px 0;
    padding: 10px;
    width: 100%;
}

button {
    cursor: pointer;
}

#respuesta {
    margin-top: 20px;
    color: blue;
}
```

PASO 5: CREAR ARCHIVO JAVASCRIPT DEL FRONTEND (SCRIPT.JS)

Archivo a crear:

```
ejercicio4-validacion-backend/public/script.js
```

Explicación detallada:

- Este archivo enviará datos del formulario al backend para que sean validados.
- Usa fetch() para interactuar con la API del servidor backend.

Ejemplo básico:

```
document.getElementById('formulario').addEventListener('submit',
function(e) {
    e.preventDefault();

    const datos = {
        nombre: document.getElementById('nombre').value,
        edad: document.getElementById('edad').value,
        email: document.getElementById('email').value
    };

    fetch('/datos', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(datos)
    })
    .then(res => res.text())
    .then(respuesta => {
        document.getElementById('respuesta').innerText = respuesta;
    })
    .catch(error => {
        document.getElementById('respuesta').innerText = "Error al
validar datos.";
```

```
        console.error(error);
    });
});
```

Resumen del porqué de cada paso en este ejercicio:

Paso	Razón principal
1	Inicializar proyecto e instalar dependencias (Express, Joi).
2	Definir servidor, lógica backend y validaciones con Joi.
3	Organizar archivos frontend separadamente del backend.
4	Crear formulario web para interacción usuario-servidor.
5	Estilizar visualmente la aplicación para mejor experiencia UX.
6	Establecer conexión dinámica frontend-backend mediante AJAX.

CONCLUSIÓN GENERAL

Este ejercicio tiene un enfoque práctico en la validación de datos en backend, algo crucial para seguridad y calidad del software:

- **Joi** facilita una validación robusta y clara.
- **Express** permite recibir y responder peticiones HTTP fácilmente.
- La estructura clara del proyecto asegura facilidad de mantenimiento y escalabilidad futura.

Realizando este ejercicio, fortalecerás tu comprensión de cómo gestionar y validar datos correctamente en una aplicación web desde el lado del servidor con Node.js.

EJERCICIO 5: VALIDACIÓN MIXTA (FRONTEND + BACKEND)

El objetivo principal de este ejercicio es combinar validaciones tanto en el **frontend** (experiencia inmediata al usuario) como en el **backend** (asegurando la integridad y seguridad).

PASO INICIAL: CONFIGURACIÓN DEL PROYECTO NODE.JS

Comandos ejecutados:

```
cd ejercicio5-validacion-mixta
npm init -y
npm install express body-parser cors joi
```

Explicación detallada:

- **cd ejercicio5-validacion-mixta**
  Nos posiciona claramente en la carpeta del proyecto, manteniendo orden y claridad al trabajar.
- **npm init -y**
  Genera automáticamente un archivo package.json. Este archivo registra las dependencias y facilita el control del proyecto.
- **npm install express body-parser cors joi**
  Instalación de dependencias importantes:
  - **Express**: Manejo simple y eficiente del servidor HTTP.
  - **body-parser**: Procesamiento de peticiones JSON.
  - **cors**: Permite interacción entre frontend y backend, incluso desde dominios diferentes.
  - **Joi**: Librería para realizar validaciones potentes y claras en el backend.

PASO 1: CREAR ARCHIVO SERVIDOR (SERVER.JS)

Archivo a crear:

```
ejercicio5-validacion-mixta/server.js
```

Explicación detallada:

- Este archivo es fundamental, pues gestionará las validaciones definitivas antes de aceptar datos.
- Se usa Joi en el backend para asegurar que los datos que pasan desde el frontend son seguros y cumplen criterios establecidos.

Ejemplo detallado del archivo (server.js):

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const Joi = require('joi');

const app = express();
```

```
app.use(bodyParser.json());
app.use(cors());
app.use(express.static('public'));

// Ruta backend para validar datos enviados desde el frontend
app.post('/validar', (req, res) => {
    const schema = Joi.object({
        usuario: Joi.string().alphanum().min(4).max(12).required(),
        contraseña: Joi.string().min(8).required(),
        email: Joi.string().email().required()
    });

    const resultado = schema.validate(req.body);
    if (resultado.error) {
        return
res.status(400).send(resultado.error.details[0].message);
    }

    res.status(200).send("Datos válidos en backend.");
});

// Inicio del servidor
app.listen(3000, () => {
    console.log("Servidor backend activo en puerto 3000");
});
```

PASO 2: CREAR CARPETA PUBLIC PARA FRONTEND

Comando ejecutado:

```
mkdir ejercicio5-validacion-mixta/public
```

Explicación detallada:

- Permite separar claramente archivos públicos estáticos (frontend) de la lógica backend, facilitando mantenimiento y organización.

PASO 3: CREAR ARCHIVO HTML (INDEX.HTML)

Archivo a crear:

```
ejercicio5-validacion-mixta/public/index.html
```

Explicación detallada:

- Proporciona la interfaz visual que interactúa directamente con el usuario.
- Contiene formulario con campos a validar tanto en frontend (rápida retroalimentación al usuario) como en backend (seguridad).

Ejemplo claro del archivo HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
    <meta charset="UTF-8">
    <title>Validación Mixta (Frontend + Backend)</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <form id="formulario">
        <input type="text" id="usuario" placeholder="Usuario (4-12
caracteres)" required>
        <input type="password" id="contraseña"
placeholder="Contraseña (mínimo 8 caracteres)" required>
        <input type="email" id="email" placeholder="Email válido"
required>
        <button type="submit">Enviar datos</button>
    </form>

    <div id="mensaje"></div>

    <script src="script.js"></script>
</body>
</html>
```

PASO 4: CREAR ARCHIVO CSS (STYLES.CSS)

Archivo a crear:

```
ejercicio5-validacion-mixta/public/styles.css
```

Explicación detallada:

- Archivo que controla diseño, presentación visual y experiencia de usuario.
- Mejora la visualización de mensajes de error o éxito tras validaciones.

Ejemplo sencillo:

```
body {
    font-family: Arial, sans-serif;
    background-color: #eef3f9;
    padding: 20px;
}

form {
    width: 300px;
    margin: auto;
}

input, button {
    width: 100%;
    padding: 8px;
    margin-top: 10px;
}

#mensaje {
    margin-top: 15px;
```

```
    text-align: center;
    font-weight: bold;
}
```

PASO 5: CREAR ARCHIVO JAVASCRIPT FRONTEND (SCRIPT.JS)

Archivo a crear:

```
ejercicio5-validacion-mixta/public/script.js
```

Explicación detallada:

- Ejecuta **validaciones rápidas** en el navegador para una respuesta inmediata al usuario.
- Envía datos al backend mediante fetch() solo cuando datos frontend sean correctos.
- Proporciona mensajes claros según resultados obtenidos.

Ejemplo concreto de validación frontend+backend:

```
document.getElementById('formulario').addEventListener('submit',
function(e){
    e.preventDefault();

    const usuario = document.getElementById('usuario').value;
    const contraseña = document.getElementById('contraseña').value;
    const email = document.getElementById('email').value;

    // Validación frontend inmediata
    if(usuario.length < 4 || usuario.length > 12){
        mostrarMensaje("Usuario debe tener entre 4 y 12
caracteres.");
        return;
    }

    if(contraseña.length < 8){
        mostrarMensaje("Contraseña mínimo 8 caracteres.");
        return;
    }

    // Enviar datos validados frontend al backend
    fetch('/validar', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({usuario, contraseña, email})
    })
    .then(res => res.text())
    .then(respuesta => mostrarMensaje(respuesta))
    .catch(err => {
        mostrarMensaje("Error en conexión con servidor backend.");
        console.error(err);
    });
});
```

```
function mostrarMensaje(msg){
    document.getElementById('mensaje').innerText = msg;
}
```

Razones del conjunto del ejercicio:

Paso	Importancia y razón
1	Configura entorno Node.js e instala librerías clave.
2	Gestiona validación robusta en backend (seguridad final).
3	Organiza archivos frontend claramente.
4	Brinda interfaz visual interactiva al usuario.
5	Mejora UX con estilos claros y visualmente atractivos.
6	Realiza validaciones inmediatas (frontend) antes del backend.

CONCLUSIÓN GENERAL DEL EJERCICIO:

Este ejercicio permite comprender cómo integrar efectivamente validaciones tanto en el frontend (para comodidad y rapidez hacia el usuario) como en el backend (garantizando seguridad e integridad de los datos recibidos).

EJERCICIO 6: GESTIÓN DE SESIONES DE USUARIO

Este ejercicio muestra cómo **implementar sesiones** para gestionar usuarios autenticados en una aplicación web utilizando **Node.js y Express**.

PASO INICIAL: PREPARACIÓN DEL ENTORNO NODE.JS PARA SESIONES

Comandos ejecutados:

```
cd ejercicio6-sesiones
npm init -y
npm install express body-parser cors express-session cookie-parser
```

Explicación detallada:

- **cd ejercicio6-sesiones**
  Nos movemos al directorio donde desarrollaremos el proyecto para mantener el orden.
- **npm init -y**
  Inicializa un proyecto Node.js y genera el archivo package.json con configuraciones predeterminadas, facilitando la gestión de dependencias.
- **npm install express body-parser cors express-session cookie-parser**
  Instalación de dependencias fundamentales:
  - **Express:** Framework web.
  - **body-parser:** Procesa datos JSON y formularios.
  - **cors:** Facilita interacciones frontend-backend.
  - **express-session:** Gestiona sesiones en el servidor.
  - **cookie-parser:** Maneja cookies para almacenar información sobre sesiones del usuario.

PASO 1: CREAR ARCHIVO SERVIDOR PRINCIPAL (SERVER.JS)

Archivo a crear:

```
ejercicio6-sesiones/server.js
```

Explicación detallada:

- El servidor configura sesiones para gestionar estados de usuarios (inicio/cierre de sesión).
- express-session y cookie-parser almacenan información en cookies seguras y manejan sesiones autenticadas.

Ejemplo detallado (server.js):

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const session = require('express-session');
const cookieParser = require('cookie-parser');
```

```
const app = express();

app.use(bodyParser.json());
app.use(cors({
    origin: 'http://localhost:3000',
    credentials: true
}));

app.use(cookieParser());
app.use(session({
    secret: 'miClaveSecreta123',
    resave: false,
    saveUninitialized: true,
    cookie: { secure: false } // false para desarrollo local (http)
}));

app.use(express.static('public'));

// Rutas básicas para login y panel
app.post('/login', (req, res) => {
    const { usuario, password } = req.body;

    if(usuario === 'usuario' && password === '1234'){
        req.session.usuario = usuario;
        res.status(200).send("Autenticado");
    } else {
        res.status(401).send("Credenciales incorrectas");
    }
});

app.get('/panel', (req, res) => {
    if(req.session.usuario){
        res.status(200).send({ usuario: req.session.usuario });
    } else {
        res.status(401).send("No autenticado");
    }
});

app.get('/logout', (req, res) => {
    req.session.destroy();
    res.status(200).send("Sesión cerrada");
});

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

PASO 2: CREAR CARPETA PÚBLICA PARA FRONTEND (PUBLIC)

Comando ejecutado:

```
mkdir ejercicio6-sesiones/public
```

Explicación detallada:

- Organización de archivos frontend (HTML, CSS, JS).
- Mejora mantenimiento y estructura del proyecto.

PASO 3: CREAR ARCHIVO HTML PARA INICIO DE SESIÓN (INDEX.HTML)

Archivo a crear:

```
ejercicio6-sesiones/public/index.html
```

Explicación detallada:

- Página que muestra formulario de login.
- Permite a usuarios autenticarse y acceder a zonas protegidas.

Ejemplo sencillo (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Iniciar Sesión</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <form id="formLogin">
        <h2>Iniciar sesión</h2>
        <input type="text" id="usuario" placeholder="Usuario"
required>
        <input type="password" id="password"
placeholder="Contraseña" required>
        <button type="submit">Entrar</button>
    </form>
    <div id="mensaje"></div>
    <script src="login.js"></script>
</body>
</html>
```

PASO 4: CREAR PÁGINA HTML PROTEGIDA PARA USUARIOS AUTENTICADOS (PANEL.HTML)

Archivo a crear:

```
ejercicio6-sesiones/public/panel.html
```

Explicación detallada:

- Página accesible exclusivamente tras iniciar sesión correctamente.
- Muestra información protegida del usuario autenticado.

Ejemplo concreto (panel.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Panel de Usuario</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h2>Panel de control</h2>
    <p id="bienvenida"></p>
    <button id="btnLogout">Cerrar sesión</button>

    <script src="panel.js"></script>
</body>
</html>
```

PASO 5: CREAR ARCHIVO CSS COMÚN (STYLES.CSS)

Archivo a crear:

```
ejercicio6-sesiones/public/styles.css
```

Explicación detallada:

- Estilos compartidos para página de inicio de sesión y panel de control.
- Facilita experiencia visual atractiva.

Ejemplo sencillo (styles.css):

```
body {
    font-family: Arial, sans-serif;
    background-color: #eef4f9;
    padding: 20px;
}

form {
    width: 250px;
    margin: auto;
}

input, button {
    display: block;
    margin-top: 10px;
    width: 100%;
    padding: 8px;
}

#mensaje {
    text-align: center;
    margin-top: 15px;
    color: red;
```

```
}

h2, #bienvenida {
    text-align: center;
}
```

PASO 6: CREAR JAVASCRIPT PARA LOGIN (LOGIN.JS)

Archivo a crear:

```
ejercicio6-sesiones/public/login.js
```

Explicación detallada:

- Envía credenciales del usuario al backend para autenticar.
- Gestiona acceso según resultado del login.

Ejemplo claro (login.js):

```
document.getElementById('formLogin').addEventListener('submit',
function(e){
    e.preventDefault();
    const usuario = document.getElementById('usuario').value;
    const password = document.getElementById('password').value;

    fetch('/login', {
        method: 'POST',
        headers: {'Content-Type':'application/json'},
        credentials: 'include',
        body: JSON.stringify({usuario, password})
    })
    .then(res => {
        if(res.ok) {
            window.location.href = 'panel.html';
        } else {
            document.getElementById('mensaje').innerText = "Login
incorrecto.";
        }
    })
    .catch(err => console.error(err));
});
```

PASO 7: CREAR JAVASCRIPT PARA PANEL DE USUARIO (PANEL.JS)

Archivo a crear:

```
ejercicio6-sesiones/public/panel.js
```

Explicación detallada:

- Verifica autenticación de usuario y muestra contenido protegido.

- Gestiona cierre de sesión.

Ejemplo claro (panel.js):

```
fetch('/panel', { credentials: 'include' })
.then(res => {
    if(res.ok) return res.json();
    else window.location.href = 'index.html';
})
.then(datos => {
    document.getElementById('bienvenida').innerText = `Bienvenido,
${datos.usuario}`;
})
.catch(() => window.location.href = 'index.html');

document.getElementById('btnLogout').addEventListener('click',
function(){
    fetch('/logout', { credentials: 'include' })
    .then(() => window.location.href = 'index.html');
});
```

CONCLUSIÓN GENERAL:

Al completar este ejercicio comprenderás claramente cómo implementar **gestión segura de sesiones de usuario**, usando herramientas modernas y seguras (express-session, cookies), con una estructura clara para la seguridad del backend y una experiencia amigable en el frontend.

EJERCICIO 7: INTEGRACIÓN CON MONGODB

En este ejercicio implementamos un **sistema completo** que valida formularios y persiste datos utilizando **MongoDB** como base de datos, usando tecnologías populares como **Express**, **Mongoose** y validaciones con **express-validator**.

PASO INICIAL: CONFIGURACIÓN DEL PROYECTO NODE.JS Y MONGODB

Comandos ejecutados:

```
cd ejercicio7-mongodb
npm init -y
npm install express body-parser cors mongoose express-validator
bcryptjs
```

Explicación detallada:

- **cd ejercicio7-mongodb**
  Movemos el terminal a la carpeta específica del ejercicio para mantener orden y claridad.
- **npm init -y**
  Inicializa el proyecto creando automáticamente package.json, esencial para gestionar dependencias.
- **npm install express body-parser cors mongoose express-validator bcryptjs**
  Instalación de dependencias necesarias:
  - **Express:** Framework backend.
  - **body-parser:** Parseo de datos JSON y formularios.
  - **cors:** Permite peticiones cross-origin frontend-backend.
  - **mongoose:** Librería que conecta y gestiona MongoDB fácilmente.
  - **express-validator:** Validaciones robustas en el backend.
  - **bcryptjs:** Librería para cifrar contraseñas de forma segura.

PASO 1: CREAR ARCHIVO SERVIDOR (SERVER.JS)

Archivo a crear:

```
ejercicio7-mongodb/server.js
```

Explicación detallada:

- Se conecta a MongoDB usando **Mongoose**.
- Gestiona rutas y validación en backend con **express-validator**.
- Protege contraseñas usando **bcryptjs**.

Ejemplo claro (server.js):

```
const express = require('express');
const mongoose = require('mongoose');
```

```
const bodyParser = require('body-parser');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const { body, validationResult } = require('express-validator');

const app = express();

mongoose.connect('mongodb://localhost:27017/miBaseDatos');

const Usuario = mongoose.model('Usuario', new mongoose.Schema({
    usuario: String,
    email: String,
    password: String
}));

app.use(bodyParser.json());
app.use(cors());
app.use(express.static('public'));

// Ruta registro de usuario
app.post('/registro', [
    body('usuario').isLength({min:4}),
    body('email').isEmail(),
    body('password').isLength({min:8})
], async (req,res)=>{
    const errores = validationResult(req);
    if(!errores.isEmpty()){
        return res.status(400).json({errores: errores.array()});
    }
    const {usuario,email,password} = req.body;
    const passwordHash = await bcrypt.hash(password, 10);
    const nuevoUsuario = new
Usuario({usuario,email,password:passwordHash});
    await nuevoUsuario.save();
    res.status(201).send("Usuario registrado correctamente");
});

// Ruta mostrar usuarios
app.get('/usuarios', async (req,res)=>{
    const usuarios = await Usuario.find({}, '-password');
    res.status(200).json(usuarios);
});

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

PASO 2: CREAR FORMULARIO HTML DE REGISTRO (INDEX.HTML)

Archivo a crear:

```
ejercicio7-mongodb/public/index.html
```

Explicación detallada:

- Proporciona interfaz para registrar usuarios.
- Facilita validaciones frontend rápidas.

Ejemplo sencillo (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Registro Usuario</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <form id="registro">
        <h2>Formulario de Registro</h2>
        <input type="text" id="usuario" placeholder="Usuario"
required>
        <input type="email" id="email" placeholder="Email"
required>
        <input type="password" id="password"
placeholder="Contraseña" required>
        <button type="submit">Registrar</button>
    </form>
    <div id="mensaje"></div>

    <script src="script.js"></script>
</body>
</html>
```

PASO 3: CREAR PÁGINA PARA MOSTRAR USUARIOS REGISTRADOS (USUARIOS.HTML)

Archivo a crear:

```
ejercicio7-mongodb/public/usuarios.html
```

Explicación detallada:

- Muestra información de usuarios registrados.
- Ofrece visibilidad a la persistencia en MongoDB.

Ejemplo sencillo (usuarios.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Usuarios Registrados</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h2>Lista de usuarios registrados</h2>
```

```
    <ul id="listaUsuarios"></ul>

    <script src="usuarios.js"></script>
</body>
</html>
```

PASO 4: CREAR ARCHIVO CSS COMÚN (STYLES.CSS)

Archivo a crear:

```
ejercicio7-mongodb/public/styles.css
```

Explicación detallada:

- Aporta diseño visual atractivo y consistente.
- Facilita interacción con formularios y listas.

Ejemplo (styles.css):

```
body {
    font-family: Arial, sans-serif;
    background-color: #f3f7fa;
    padding: 20px;
}

form, ul {
    width: 300px;
    margin: auto;
}

input, button {
    display: block;
    padding: 8px;
    width: 100%;
    margin-top: 10px;
}

#mensaje {
    text-align: center;
    color: green;
    margin-top: 10px;
}
```

PASO 5: CREAR JAVASCRIPT PARA FORMULARIO REGISTRO (SCRIPT.JS)

Archivo a crear:

```
ejercicio7-mongodb/public/script.js
```

Explicación detallada:

- Valida frontend y envía datos a backend.
- Muestra respuestas claras del servidor.

Ejemplo claro (script.js):

```
document.getElementById('registro').addEventListener('submit',
function(e){
    e.preventDefault();
    const datos = {
        usuario: document.getElementById('usuario').value,
        email: document.getElementById('email').value,
        password: document.getElementById('password').value
    };

    fetch('/registro', {
        method:'POST',
        headers:{'Content-Type':'application/json'},
        body: JSON.stringify(datos)
    })
    .then(res => res.text())
    .then(mensaje => {
        document.getElementById('mensaje').innerText = mensaje;
    })
    .catch(err => console.error(err));
});
```

PASO 6: CREAR JAVASCRIPT PARA MOSTRAR LISTA DE USUARIOS (USUARIOS.JS)

Archivo a crear:

```
ejercicio7-mongodb/public/usuarios.js
```

Explicación detallada:

- Carga dinámicamente usuarios desde MongoDB mediante backend.
- Permite visualizar claramente datos almacenados.

Ejemplo sencillo (usuarios.js):

```
fetch('/usuarios')
.then(res => res.json())
.then(usuarios => {
    const lista = document.getElementById('listaUsuarios');
    usuarios.forEach(u => {
        const li = document.createElement('li');
        li.textContent = `${u.usuario} (${u.email})`;
        lista.appendChild(li);
    });
})
.catch(err => console.error(err));
```

CONCLUSIÓN GENERAL:

Este ejercicio permite desarrollar una solución completa que combina validaciones robustas, cifrado de datos sensibles (contraseñas) y almacenamiento en MongoDB mediante Mongoose, dando como resultado una aplicación web moderna, segura y altamente funcional.

CONCLUSION

Con estos ejercicios, hemos creado un completo conjunto de ejemplos de validación de formularios que cubren:

1. Validación básica del lado del cliente
2. Validación avanzada del lado del cliente con información en tiempo real
3. Validación del lado del servidor con Node.js y Express
4. Validación combinada de frontend y backend
5. Gestión de sesiones para formularios autenticados
6. Integración con MongoDB para la persistencia de datos

Estos ejemplos demuestran diferentes técnicas para validar la entrada del usuario, proporcionar retroalimentación y asegurar la integridad de los datos. La progresión de escenarios de validación simples a complejos proporciona una base sólida para implementar la validación de formularios en aplicaciones del mundo real.

Para ejecutar cada ejercicio, necesitarás:

1. Navegue hasta el directorio de ejercicios
2. Instale las dependencias necesarias con npm install
3. Inicie el servidor con node server.js
4. Accede a la aplicación en tu navegador en http://localhost:3000

Para el ejercicio de MongoDB, necesitarás tener MongoDB instalado y funcionando en tu sistema. Si no tienes MongoDB instalado, puedes modificar la cadena de conexión en el archivo server.js para utilizar una base de datos en la nube MongoDB Atlas en su lugar.