📖Biblia Definitiva para Desarrollar una SPA con Login, sin Frameworks

🧰Introducción

Esta es una guía paso a paso para crear una Single Page Application (SPA) desde cero usando **HTML, CSS y JavaScript Vanilla** (sin frameworks ni librerías como React, Bootstrap o jQuery). Incluye autenticación, CRUD, json-server, almacenamiento local, rutas personalizadas y buenas prácticas. Ideal para principiantes, gente con ansiedad pre-examen y devs autodidactas.

🔗1. Setup del Proyecto con npm

1.1 Crea una carpeta de proyecto

```
mkdir admin-cursos && cd admin-cursos
```

1.2 Inicializa npm

```
npm init -y
```

1.3 Instala json-server (simula tu backend)

```
npm install json-server --save-dev
```

1.4 Agrega el script en `package.json`

```
"scripts": {
  "server": "json-server --watch db.json --port 3000"
}
```

1.5 Crea `db.json`

Este será tu backend fake:

```
{
  "users": [
    {
```

```
      "id": 1,
      "name": "Admin",
      "email": "admin@admin.com",
      "password": "admin123",
      "role": "admin"
    }
  ],
  "courses": [],
  "enrollments": []
}
```

1.6 Ejecuta el servidor

```
npm run server
```

Esto inicia la API REST en `http://localhost:3000`

🧱 2. Estructura del Proyecto

```
admin-cursos/
├── db.json
├── package.json
├── /src
│   ├── main.js
│   ├── /assets
│   ├── /components
│   │   ├── header.js
│   │   ├── sidebar.js
│   ├── /pages
│   │   ├── login.html
│   │   ├── register.html
│   │   ├── dashboard.html
│   │   └── public.html
│   ├── /services
│   │   ├── auth.js
│   │   ├── users.js
│   │   └── courses.js
│   └── /utils
│       ├── storage.js
│       └── validation.js
```

🔐3. Módulo de Autenticación

3.1 Registro

En `auth.js`:

```
export async function registerUser(userData) {
  const res = await fetch("http://localhost:3000/users", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(userData)
  });
  return res.json();
}
```

3.2 Login

```
export async function loginUser(email, password) {
  const res = await fetch(`http://localhost:3000/users?email=${email}&password=$
{password}`);
  const data = await res.json();
  return data.length ? data[0] : null;
}
```

3.3 Guardar sesión

```
localStorage.setItem("user", JSON.stringify(usuario));
```

3.4 Proteger rutas

```
const user = JSON.parse(localStorage.getItem("user"));
if (!user || user.role !== "admin") {
  location.href = "login.html";
}
```

⚙️4. CRUD con Fetch y json-server

4.1 Obtener datos (GET)

```
const res = await fetch("http://localhost:3000/courses");
const cursos = await res.json();
```

4.2 Crear (POST)

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

4.3 Editar (PUT)

```
await fetch(`http://localhost:3000/courses/${id}`, {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(cursoActualizado)
});
```

4.4 Eliminar (DELETE)

```
await fetch(`http://localhost:3000/courses/${id}`, {
  method: "DELETE"
});
```

---

5. SPA: Navegación sin recargar

5.1 HTML con data-link

```
<a href="/dashboard" data-link>Dashboard</a>
```

5.2 JS: interceptar navegación

```
document.body.addEventListener("click", e => {
  if (e.target.matches("[data-link]")) {
```

```
    e.preventDefault();
    navigateTo(e.target.href);
  }
});
```

5.3 Navegar y renderizar

```
function navigateTo(url) {
  history.pushState(null, null, url);
  renderContent();
}

function renderContent() {
  const path = location.pathname;
  const view = routes[path] || "404.html";
  fetch(view).then(res => res.text()).then(html => {
    document.getElementById("app").innerHTML = html;
  });
}
```

🧾6. localStorage y sessionStorage

6.1 Guardar sesión

```
localStorage.setItem("user", JSON.stringify(usuario));
```

6.2 Recuperar

```
const usuario = JSON.parse(localStorage.getItem("user"));
```

6.3 Borrar sesión

```
localStorage.removeItem("user");
```

📏7. Buenas Prácticas y Evaluación

🔧camelCase en todo

- `nombreUsuario`, `idCurso`, `formularioLogin`

🔧Código Modular

- Una función = una tarea.
- Archivos por responsabilidad (`auth.js`, `users.js`, etc.)

🔧Comentarios útiles

```
// Esta función elimina un curso por ID\async function eliminarCurso(id) {...}
```

🔧HTML semántico y accesible

- Usa etiquetas como `<main>`, `<section>`, `<form>` correctamente.

🔗RESUMEN RÁPIDO PARA EXAMEN

Tema	¿Qué recordar?
npm	`npm init`, `npm run server`
json-server	Rutas REST: `GET`, `POST`, `PUT`, `DELETE`
SPA	`pushState`, `renderContent`, `routes`
login	`fetch` + `localStorage`
rutas protegidas	Validar rol antes de mostrar página
CRUD	Separar funciones por acción, usar async/await
Modularización	Archivos por tipo de lógica
UI/UX	CSS con Flex/Grid, sin librerías

¿Listo para romperla? Si estás estresado, respira hondo. Esta guía es tu mapa. Te cubre desde cero hasta lo más complejo, paso a paso y con sentido práctico.