✅ Estructura de evaluación del examen (según los criterios proporcionados)

---

1. 🧠 Nomenclatura (camelCase)

- ✅ Toda variable debe estar en **camelCase**.
    - Ejemplos correctos: `nombreUsuario`, `idProducto`, `listaTareas`.
    - ⚠️ **Pérdida de puntos** si usas `snake_case`, `PascalCase` o mayúsculas innecesarias.

📌 **Consejo útil:**

- Usa nombres descriptivos pero breves:
  ✔️ `botonGuardar`, `formularioLogin`, `datosUsuario`

---

2. 🖥️ Servidor local: JSON-SERVER corriendo

- Debes levantar el servidor con `json-server --watch db.json --port 3000`.
- La API debe estar funcional (`GET`, `POST`, `PUT`, `DELETE`) para trabajar con `fetch`.

📌 **Consejo útil:**

- Verifica antes del examen que el archivo `db.json` esté bien formado (no haya comas sobrantes).
- Usa `http://localhost:3000/recursos` en tus fetch.

---

3. 🛠️ CRUD con JavaScript y Fetch

- Implementa las 4 operaciones:
    - `GET`: mostrar datos

- `POST`: agregar datos
- `PUT/PATCH`: editar datos
- `DELETE`: eliminar datos

📌 **Buenas prácticas:**

- Usa funciones separadas por operación (modulariza).
- Muestra mensajes de éxito/error al usuario.
- Recarga la lista después de cada acción (sin recargar la página entera).
- Usa `async/await` para claridad.

---

4. 🧾 Replicación de plantilla Figma

- Debes respetar:
  - **Estructura HTML** visual (encabezados, botones, inputs, cards…)
  - **Estilos (CSS)** similares: fuentes, colores, márgenes, disposición
  - **Comportamientos interactivos** si están definidos (hover, clicks…)

📌 **Consejos útiles:**

- Usa `class` bien nombradas.
- Sigue el orden jerárquico del diseño (encabezados, contenedores, listas…).
- Evita sobrecargar el HTML con estilos inline; usa CSS externo o interno.

---

5. 📄 SPA (Single Page Application)

- Implementa navegación **sin recargar la página**.

- Al hacer click en un enlace, se debe cambiar el contenido con `history.pushState` y `renderContent`.

📌 **Consejos útiles:**

- Usa `data-link` en los `<a>` para interceptar clicks.
- Crea una función `navigateTo()` y una `renderContent()` clara y comentada.
- Usa un objeto `routes` para mapear rutas a contenido o funciones.

---

6. 💾 Uso de `localStorage` y `sessionStorage`

- Debes implementar al menos uno de cada:
    - `localStorage`: para guardar info persistente (ej. nombre de usuario, token).
    - `sessionStorage`: para guardar algo temporal (sesión activa, navegación).

📌 **Consejos útiles:**

- Usa `JSON.stringify()` para guardar objetos y `JSON.parse()` al recuperar.
- Borra los datos con `removeItem` en logout.
- Usa valores guardados para mostrar info o restringir acceso.

---

7. ⚙️ Funcionalidad completa del script JS

- El JS debe ser funcional e integral:
    - Todos los botones deben responder.
    - No deben haber errores en consola.
    - Las vistas deben actualizarse correctamente.
    - Las operaciones CRUD deben reflejarse visualmente.

📌 **Consejos útiles:**

- Haz validaciones antes de enviar formularios.
- Muestra mensajes o estados (“Guardando...”, “Usuario creado”, etc).
- Usa `console.log()` solo para pruebas, quítalos al final.

---

8. 🧩 Modularización del código

- El código debe estar **organizado en funciones reutilizables**.
- Idealmente separa por archivo si te lo permiten (`main.js`, `api.js`, `dom.js`, etc).

📌 **Consejos útiles:**

- Una función = una responsabilidad clara.
  ✔️ `crearUsuario()`, `eliminarProducto(id)`, `cargarVista(ruta)`
- Evita duplicar código. Si haces `.fetch()` 3 veces, usa una función común.

---

9. 💬 Comentarios explicativos

- Cada función o bloque importante debe tener un comentario breve pero claro.

📌 **Consejos útiles:**

Usa comentarios así:

```
 js
CopyEdit
// Esta función elimina un producto del servidor y actualiza la
vista
async function eliminarProducto(id) {
  ...
}
```

-

- No comentes lo obvio (como `let x = 10 // se crea x con valor 10`), pero sí la intención del código.

⭐ Resumen visual de los puntos a evaluar

Categoría	¿Qué evalúan?	Puntos clave
camelCase	Nombrar bien variables	nombres descriptivos en camelCase
json-server	Que el backend esté funcional	`db.json`, rutas válidas
CRUD con fetch	Todas las operaciones básicas	modularidad, mensajes, sin recargar
Figma	HTML/CSS fiel a la plantilla	jerarquía, clases, estilos limpios
SPA	Navegación sin recarga	`pushState`, `renderContent`, `routes`
local/sessionStorage	Uso correcto de ambos	persistencia, control de sesión
Funcionalidad global	Todo el JS funcione sin errores	sin fallos, sin consola rota
Modularización	Código bien separado	funciones limpias, sin repetir
Comentarios	Funciones y lógicas bien explicadas	comentarios útiles y claros