Adaptar projeto para JS moderno

Resumo direto e prático

O que levar em conta ao adaptar um projeto JavaScript para ter um ponto de entrada `main` organizado e escalável:

🎯 Objetivo

Centralizar a inicialização da aplicação em um **ponto de entrada (`main.js`)**, deixando as páginas e módulos mais limpos, reutilizáveis e fáceis de manter.

⚙️ Estrutura atual

```
/vitrine-ml5/mlab5
 /assets
 /controle
   /gerador
   /index.html
 /css
 /cursos
   /css3
     /01
       /index.html
     /02
       /index.html
     /…
     /index.html
   /html5
     /01
       /index.html
     /02
       /index.html
     /…
     /index.html
  /...
 /images
 /navegacao
 /cursos
 /playground
 /projetos
   /lousa
 /js
   /ui
     /components
   main.js
index.html
```

⚙️ Estrutura básica recomendada

```
/src
  /components
  /pages
  /utils
  main.js
index.html
```

- **`main.js`**: ponto de entrada — carrega dependências, inicializa o app e controla o fluxo principal.
- **`/components`**: módulos reutilizáveis (funções, classes, ou componentes).
- **`/pages`**: código específico de cada página (scripts, controladores).
- **`/utils`**: funções auxiliares genéricas.

🧩 1. Extraia scripts inline para arquivos `.js`

Evite `<script>` com lógica dentro do HTML.
Em vez disso, use:

```
<script type="module" src="./src/main.js"></script>
```

E no `main.js`:

```
import initHome from './pages/home.js';
import initLogin from './pages/login.js';

if (window.location.pathname === '/home') initHome();
else if (window.location.pathname === '/login') initLogin();
```

🔄 2. Use módulos ES6 (`import` / `export`)

- **Exportar funções ou objetos** que cada parte do projeto precisa expor.
- **Importar** só o que for necessário no `main` ou em outros módulos.

```
// utils/dom.js
export function qs(selector) {
  return document.querySelector(selector);
}
----------x----------
// pages/home.js
import { qs } from '../utils/dom.js';

export default function initHome() {
  const button = qs('#btn');
  button.addEventListener('click', () => console.log('Home ready'));
}
```

🧱 3. Modularize lógica de inicialização

Cada módulo/página deve cuidar **só da própria inicialização** (event listeners, render, etc.).
O `main.js` apenas decide **o que carregar**.

🧰 4. Ferramentas úteis

- Se o projeto crescer, use **Vite**, **Webpack**, ou **Parcel** para empacotar tudo e gerar um único bundle.
- Isso permite usar importações relativas, assets otimizados e live reload.

🔍 5. Boas práticas extras

- Prefira **nomes explícitos** (`initDashboard()`, não `loadPage()`).
- Organize imports de forma consistente (utils → components → pages).
- Evite dependências globais (como variáveis no `window`).
- Use **`DOMContentLoaded`** para garantir que o DOM esteja pronto antes da inicialização.

✅ Em resumo:

1. Crie um **arquivo `main.js`** como ponto de entrada.
2. Mova scripts inline para **arquivos separados**.
3. Use **módulos ES6** para importar/exportar.
4. Estruture por **função e responsabilidade** (não por tipo de arquivo).
5. Adote um **bundler** se o projeto crescer.