**Explicación del JavaScript (línea por línea y en cristiano)**

// assets/js/i18n-inline.js

Solo es un **comentario** para saber en qué archivo estás si mirás la consola del navegador.

(() => {

Esto es una **IIFE** (Immediately Invoked Function Expression).  
Traducción: es una función que **se ejecuta apenas se carga** el archivo y que **encierra** todo el código para que **no contamine** el espacio global (window).  
Beneficio: si más adelante agregás otros scripts, no se pisan nombres de variables/funciones.

// Utilidad: acceso a claves anidadas tipo "hero.hi"

const getByPath = (obj, path) =>

path.split('.').reduce((o, k) => (o && k in o) ? o[k] : undefined, obj);

**¿Qué hace?**

* getByPath es una **función ayudante** para leer un valor dentro de un objeto **anidado** usando una **ruta en texto** (por ejemplo "hero.hi").
* path.split('.') convierte "hero.hi" en ["hero","hi"].
* reduce va entrando **clave por clave**: obj["hero"] → luego ["hi"].

**¿Por qué así?**

Sin esto tendrías que escribir obj && obj.hero && obj.hero.hi cada vez. Con getByPath(dict, "hero.hi") queda **limpio y reutilizable**.

// Diccionario ES/EN/IT/PT (dejado tal cual)

const I18N = {

es: { ... },

en: { ... },

it: { ... },

pt: { ... }

};

**¿Qué hace?**

* I18N es tu **diccionario de traducciones**.
* Tiene 4 idiomas de primer nivel (es, en, it, pt) y adentro más objetos (nav, hero, projects, etc.) con los textos.

**¿Por qué así?**

* Mantiene **todo el contenido** (los textos) separado del **HTML**.
* Tus elementos del HTML dicen “quiero el texto hero.hi” y acá está el valor real.

Nota: en tu versión original en inglés tenías la clave senda **duplicada**. En JS, la **segunda** definición **pisa** la primera. No rompe la app, pero conviene dejar una sola.

function applyTranslations(lang) {

const dict = I18N[lang] || I18N.es;

document.documentElement.setAttribute('lang', lang);

**¿Qué hace?**

* applyTranslations es la **función principal** que **pinta** la página en el idioma elegido.
* dict elige el objeto del idioma. Si no existe (ej. pusiste "fr"), **cae a español**.
* Cambia el atributo lang en <html> (útil para **accesibilidad y SEO**).

// Elementos con texto puro

document.querySelectorAll('[data-i18n]').forEach(el => {

const key = el.getAttribute('data-i18n');

const value = getByPath(dict, key);

if (typeof value === 'string') {

// Si el texto contiene HTML (strong, etc.) usamos innerHTML

if (value.includes('<') && value.includes('>')) {

el.innerHTML = value;

} else {

el.textContent = value;

}

}

});

**¿Qué hace?**

* Busca **todos** los elementos del HTML que tienen el atributo data-i18n.
* Para cada uno:
  + Lee la **clave** (por ejemplo data-i18n="hero.hi").
  + Busca el texto en el diccionario con getByPath.
  + Si encuentra un **string**:
    - Si el texto **contiene etiquetas** (ej. <strong>), usa innerHTML (para renderizar esas etiquetas).
    - Si no, usa textContent (más **seguro** porque no interpreta HTML).

**¿Por qué así?**

* data-i18n es como un **“gancho”** que conecta tu HTML con el texto del diccionario.
* innerHTML solo cuando **vos** pusiste HTML en el diccionario (como <strong>).  
  Para todo lo demás, textContent evita problemas de seguridad/errores.

// Guarda preferencia y sincroniza ?lang

try { localStorage.setItem('lang', lang); } catch {}

const url = new URL(window.location);

url.searchParams.set('lang', lang);

window.history.replaceState({}, '', url);

}

**¿Qué hace?**

* **Guarda** el idioma preferido del usuario en localStorage (si el navegador lo permite).
* Actualiza la URL con ?lang=xx **sin recargar** (History API).  
  Ejemplo: si elegís inglés, la URL queda ...?lang=en.

**¿Por qué así?**

* localStorage: si el usuario vuelve, mantiene su idioma.
* ?lang=xx: podés **compartir** la URL y la otra persona lo ve **directamente** en ese idioma.

function detectLanguage() {

const fromUrl = new URLSearchParams(location.search).get('lang');

if (fromUrl && I18N[fromUrl]) return fromUrl;

**¿Qué hace?**

* Decide cuál es el **idioma inicial**.
* **Primero**, si la URL trae ?lang=xx y ese idioma existe en I18N, usa ese.

let stored;

try { stored = localStorage.getItem('lang'); } catch {}

if (stored && I18N[stored]) return stored;

**¿Qué hace?**

* **Segundo**, si hay un idioma guardado en localStorage y es válido, usa ese.

const browser = (navigator.language || 'es').slice(0,2);

return I18N[browser] ? browser : 'es';

}

**¿Qué hace?**

* **Tercero**, usa el idioma del **navegador** (por ejemplo en-US → en).
* Si no está soportado, **vuelve a español**.

**¿Por qué este orden?**

1. URL (explícito y compartible)
2. Recuerdo del usuario
3. Preferencia del navegador
4. Fallback que **siempre funciona**: es

// Init al cargar el DOM

document.addEventListener('DOMContentLoaded', () => {

const langSelect = document.getElementById('langSwitcher');

const initialLang = detectLanguage();

if (langSelect) langSelect.value = initialLang;

applyTranslations(initialLang);

**¿Qué hace?**

* Espera a que el **HTML esté listo** (para que exista el select y todos los elementos con data-i18n).
* Calcula el **idioma inicial** con la lógica anterior.
* Si existe el <select> de idiomas, lo **sincroniza** para que muestre ese idioma.
* Llama a applyTranslations(initialLang) para **pintar** los textos.

if (langSelect) {

langSelect.addEventListener('change', (e) => applyTranslations(e.target.value));

}

});

**¿Qué hace?**

* Cuando el usuario **cambia** el idioma en el <select>, vuelve a:
  + Pintar los textos,
  + Guardar el idioma,
  + Actualizar la URL.

})();

Cierra la IIFE.  
Resultado: **todo** el código queda ordenado, **no hay variables globales**, y el script corre **solo**.

**Informe resumido (qué hace y por qué)**

* **Qué resuelve este JS**
  1. Mantiene un **diccionario** de textos por idioma.
  2. Detecta el **idioma inicial** (URL > guardado > navegador > ES).
  3. **Rellena** el HTML: busca elementos con data-i18n="ruta.de.clave" y les pone el texto correspondiente.
  4. Si el texto tiene **HTML (ej. <strong>)**, lo respeta con innerHTML.
  5. Guarda la elección en localStorage y actualiza la URL con ?lang=xx sin recargar.
* **Por qué está armado así**
  1. La **IIFE** aísla tu código y evita conflictos con otros scripts.
  2. getByPath simplifica **muchísimo** leer propiedades anidadas ("projects.adventure.b2").
  3. applyTranslations es el **centro**: actualiza textos, lang de <html>, localStorage y ?lang.
  4. DOMContentLoaded garantiza que el DOM ya existe antes de tocarlo.
  5. El orden de detección de idioma prioriza lo **explícito** y ofrece **fallback** seguro.
* **Cosas a cuidar**
  1. **Rutas con mayúsculas/minúsculas**: en GitHub Pages, Imagenes ≠ Images.
  2. **IDs duplicados** (id="proyectos") rompen el scroll y pueden confundir.
  3. **Clave duplicada** senda en inglés: dejá solo una.
  4. Si el <select id="langSwitcher"> no existe, el código igual funciona (el if (langSelect) evita errores).

Si querés, te armo una variante **igual** a esta pero con los textos en **JSON separados** (uno por idioma) y el loader que los trae con fetch, para que tu JS quede bien liviano.