Perfeito — abaixo está a explicação detalhada, linha a linha, do seu arquivo script.js. Mantive a ordem original e expliquei o propósito e comportamento de cada trecho/linha, com notas sobre comportamentos implícitos, possíveis melhorias e riscos. Se quiser, posso aplicar melhorias automáticas depois da revisão (por exemplo validação de e-mail, substituição de alert por UI, envio via API, usabilidade, testes).

Arquivo: script.js

Descrição geral: módulo auto-executável (IIFE) que inicializa a internacionalização (i18n) por dicionário, controla seleção de idioma, adiciona smooth scroll para links, aplica comportamento de header sticky com sombra, animação de carregamento da imagem hero, revela seções com IntersectionObserver e implementa envio de contato via mailto e WhatsApp.

Explicação linha a linha (comentários e contextos):

// script.js

Comentário inicial indicando o nome do arquivo.

(function(){

Início de uma IIFE (Immediately Invoked Function Expression). Isola variáveis no escopo local para evitar poluir o escopo global.

'use strict';

Habilita o modo estrito do JavaScript, que previne algumas práticas inseguras (por exemplo, variáveis globais acidentais) e lança erros em situações silenciosas.

5-... Translations dictionary

Declaração da constante translations (objeto), que contém dicionários de texto para vários idiomas: pt, en, fr, es, zh.

Cada chave (por exemplo pt) mapeia várias chaves de interface para textos localizados, por exemplo:

title, slogan, btn\_servicos, btn\_contact, etc.

Uso: o dicionário é lido por applyTranslations para preencher elementos com atributos data-i18n e data-i18n-placeholder.

Observações:

O dicionário é carregado inteiro no front; em projetos maiores, melhor carregar dinamicamente apenas o idioma necessário.

Os textos incluem caracteres especiais e aspas tipográficas — tudo válido, mas cuidado com codificação em arquivos salvo em UTF-8.

(Depois do dicionário)

... // Apply translations to page

Função applyTranslations(lang) definida:

const dict = translations[lang] || translations.pt;

Seleciona o dicionário para o idioma pedido; se não existir, usa o português como fallback.

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

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

if(key && dict[key]) el.textContent = dict[key];

});

Seleciona todos os elementos com atributo data-i18n.

Para cada elemento: obtém a chave (ex.: 'title') e, se existir esse par no dicionário, define o texto do elemento (textContent) para o valor traduzido.

Observações:

textContent sobrescreve todo o conteúdo textual do elemento — cuidado se o elemento contiver elementos filhos que você queira preservar.

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

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

if(key && dict[key]) el.setAttribute('placeholder', dict[key]);

});

Faz o mesmo para placeholders de inputs/textarea usando data-i18n-placeholder.

document.documentElement.lang = (lang === 'zh' ? 'zh' : lang);

Define o atributo lang do elemento <html>. Usa 'zh' para chinês; caso contrário, usa o valor de lang passado.

Observação: isso melhora SEO e acessibilidade (leitores de tela).

localStorage.setItem('mv\_lang', lang);

Persiste a escolha do idioma no localStorage com chave 'mv\_lang', para lembrar a preferência do usuário.

... // Init language selector

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

Obtém o elemento <select id="langSelect"> no DOM (controle de idioma).

Se esse elemento não existir, o código atual lançaria erro quando tentar adicionar eventListener. (No seu HTML existe esse elemento.)

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

Adiciona um listener que chama applyTranslations quando o usuário muda o idioma no select. Passa o value selecionado.

const initLang = localStorage.getItem('mv\_lang') || 'pt';

Lê a preferência de idioma previamente armazenada no localStorage. Se não houver, usa 'pt' como padrão.

langSelect.value = initLang;

Atualiza o select para o valor inicial.

applyTranslations(initLang);

Aplica as traduções imediatamente para o idioma inicial (preenche a página).

... // Smooth scroll

document.querySelectorAll('[data-scroll]').forEach(link=>{

link.addEventListener('click', function(e){

e.preventDefault();

const href = this.getAttribute('href');

const target = document.querySelector(href);

if(target){

const top = target.getBoundingClientRect().top + window.pageYOffset - document.querySelector('.site-header').offsetHeight;

window.scrollTo({top, behavior:'smooth'});

}

});

});

Para cada link com data-scroll:

Intercepta o clique (e.preventDefault()) para impedir comportamento padrão (navegar instantaneamente).

Lê o href (por exemplo #services) e seleciona o elemento alvo.

Se existir, calcula a posição vertical do alvo em relação ao documento:

target.getBoundingClientRect().top = distância entre topo da viewport e o elemento.

window.pageYOffset = deslocamento atual da página.

subtrai a altura da .site-header para evitar que o header sobreponha o conteúdo ao rolar.

window.scrollTo({top, behavior:'smooth'});

Executa rolagem suave até o topo calculado.

Observações:

Suporte de navegadores modernos é bom para scroll behavior.

Se document.querySelector('.site-header') retornar null, isso geraria erro ao calcular offsetHeight — porém o HTML tem o header.

... // Sticky header shadow on scroll

const header = document.querySelector('.site-header');

Captura o elemento do cabeçalho.

window.addEventListener('scroll', ()=>{

if(window.scrollY > 20) header.classList.add('scrolled'); else header.classList.remove('scrolled');

});

Ao rolar a janela:

Se a posição vertical (scrollY) for maior que 20px, adiciona a classe 'scrolled' ao header.

Caso contrário, remove a classe.

Observações:

A classe 'scrolled' está estilizada no CSS para adicionar sombra/borda, dando feedback visual de estado sticky.

Se header for null (improvável), chamar header.classList jogaria erro — poderia adicionar checagem if(header) ....

... // Hero image fade-in

const heroImg = document.querySelector('.hero-image');

Seleciona a imagem com classe .hero-image.

if(heroImg){

heroImg.addEventListener('load', ()=>{ heroImg.style.opacity = '1'; heroImg.style.transform = 'translateY(0)'; });

if(heroImg.complete){ heroImg.style.opacity='1'; heroImg.style.transform='translateY(0)'; }

}

Se encontrou a imagem, adiciona um listener load que, quando a imagem estiver carregada, aplica estilos inline para opacidade e transform (faz o fade-in/slide).

Em seguida, checa heroImg.complete (caso a imagem já esteja em cache/ carregada) e aplica os mesmos estilos imediatamente.

Observações:

Isso complementa a transição definida em CSS para .hero-image (transition).

Bom para UX; se quisermos remover estilos inline mais tarde, poderia usar classes CSS ao invés de inline.

... // Reveal on scroll with IntersectionObserver

const reveals = document.querySelectorAll('.fade-slide');

Seleciona todas as seções com classe .fade-slide.

const io = new IntersectionObserver((entries)=>{

entries.forEach(entry=>{

if(entry.isIntersecting) entry.target.classList.add('visible');

});

},{threshold:0.12});

Cria uma instância de IntersectionObserver:

Callback recebe entries; para cada entry (uma observação de elemento), se entry.isIntersecting (elemento entrou na viewport), adiciona a classe 'visible' ao elemento alvo.

Threshold 0.12 => dispara quando 12% do elemento estiver visível.

reveals.forEach(r=> io.observe(r));

Registra cada elemento .fade-slide para observação.

Observações:

A classe .visible remove opacidade/transform e a transição definida em CSS faz a animação de entrada.

Não há limpeza (unobserve) se o elemento já estiver visível — isso é ok, já que adiciona a classe uma vez. Para performance, poderia unobserve quando tiver sido revelado.

... // Contact: mailto and whatsapp send

const emailBtn = document.getElementById('emailSendBtn');

const waBtn = document.getElementById('waSendBtn');

Seleciona os botões do formulário.

function buildMailto(name,email,message){

const subject = encodeURIComponent('Contacto MVUMBISI - ' + (name||'Novo Contato'));

const body = encodeURIComponent('Nome: ' + (name||'') + '\nEmail: ' + (email||'') + '\n\nMensagem:\n' + (message||''));

return 'mailto:sipevima17@hotmail.com?subject=' + subject + '&body=' + body;

}

Função que constrói uma URI mailto: com subject e body codificados.

Observações:

Usa encodeURIComponent para escapar caracteres especiais.

Mailto tem limites de tamanho do URL; mensagens muito grandes podem não caber.

Endereço de destino sipevima17@hotmail.com está codificado na função — se for mudar, atualize aqui.

function buildWaLink(name,email,message){

const text = encodeURIComponent('Olá MVUMBISI TECH ACADEMY, tenho um contacto. Nome: ' + (name||'') + ' Email: ' + (email||'') + ' Mensagem: ' + (message||''));

return 'https://wa.me/244937663075?text=' + text;

}

Função que gera link de WhatsApp via API de link direto (wa.me). O número +244937663075 é embutido.

Observações:

Esse link abre o WhatsApp (web/cliente), dependendo do dispositivo.

Também tem limite de comprimento.

if(emailBtn){

emailBtn.addEventListener('click', ()=>{

const name = document.getElementById('name').value.trim();

const email = document.getElementById('email').value.trim();

const message = document.getElementById('message').value.trim();

if(!name||!email||!message){ alert('Por favor preencha todos os campos.'); return; }

window.location.href = buildMailto(name,email,message);

});

}

Se existir o botão de e-mail:

Ao clicar:

Lê os valores dos inputs #name, #email, #message e .trim() para remover espaços em branco.

Valida presença (nome, email, mensagem). Se faltar, mostra um alert (nativo) e interrompe.

Se tudo ok, direciona o navegador para a URL mailto: (abre cliente de email do usuário).

Observações / Melhorias:

Validação de formato de email (regex) seria útil.

Substituir alert por UI inline melhora experiência.

Considerar envio via API para rastrear leads (ver recomendação anterior).

if(waBtn){

waBtn.addEventListener('click', ()=>{

const name = document.getElementById('name').value.trim();

const email = document.getElementById('email').value.trim();

const message = document.getElementById('message').value.trim();

if(!name||!email||!message){ alert('Por favor preencha todos os campos.'); return; }

window.open(buildWaLink(name,email,message), '\_blank');

});

}

Comportamento similar ao botão de e-mail, mas abre um window.open com o link do WhatsApp em nova aba/janela.

Observação: em dispositivos móveis, abrirá o app WhatsApp se instalado.

... final

})();

Fecha a IIFE e a executa imediatamente. Todo o código acima roda assim que o script é carregado.

Notas gerais, problemas e melhorias recomendadas (com prioridades rápidas):

Segurança & robustez

Checar existência de elementos antes de usar (por exemplo if(langSelect) { ... }) para evitar erros em situações onde o HTML seja modificado.

Substituir alert por um modal / mensagem inline (UX).

Sanitizar entradas se enviar para backend.

Validação de e-mail

Atualmente só valida presença. Pode validar o formato do e-mail com regex simples antes de enviar.

Envios de formulário

Mailto/wa limitação: não grava leads nem permite confirmação backend.

Recomendação: usar fetch para um endpoint (serverless) ou serviço Formspree/Netlify Forms. Posso implementar um exemplo com Formspree (front-end) agora.

I18n

O dicionário está todo no cliente — para páginas grandes, carregar por demanda.

Use aria-lang e hreflang para SEO (adicionar <link rel="alternate" hreflang="..."> no HTML se tiver versões em outros idiomas).

Performance

Minificar JS em produção.

Se aumentar a lógica, modularizar (ES modules) e carregar scripts com defer.

Acessibilidade

Labels nos inputs: no HTML atual, inputs não têm labels; adicione <label for="name">Nome</label> para melhorar.

Mensagens de erro acessíveis com aria-live.

Testes

Adicionar simples testes unitários para funções utilitárias (ex.: buildMailto, buildWaLink) se houver pipeline de CI.

Quer que eu:

gere uma versão do script.js com pequenas melhorias automáticas (checagens de existência dos elementos, validação de email, substituir alert por mensagem inline)? ou

implemente envio via Formspree (front-end change no index.html + fetch no script.js) para começar a capturar leads? ou

adicione labels e mensagens de erro acessíveis ao formulário (HTML + CSS + JS)?

Indique a opção (1, 2 ou 3) ou peça outra alteração e eu aplico as mudanças agora no seu repositório.