# 1

# Implémentation des fonctionnalités interactives en JavaScript

JavaScript est le langage de programmation qui permet de rendre un site web interactif et dynamique. Dans ce chapitre, nous présentons les fonctionnalités JavaScript que nous avons développées pour le site ÉcoTourisme Maroc. En tant que débutants en développement web, nous avons créé deux scripts principaux : `main.js` pour les interactions générales du site et `contact.js` pour la validation du formulaire de contact.

## 1.1. Introduction à JavaScript dans notre projet

JavaScript est un langage de programmation côté client qui s'exécute directement dans le navigateur de l'utilisateur. Il permet de :

- Réagir aux actions de l'utilisateur (clics, survol, saisie)
- Modifier dynamiquement le contenu des pages
- Valider les formulaires avant envoi
- Créer des animations et effets visuels
- Améliorer l'expérience utilisateur globale

### 1.1.1. Organisation des fichiers JavaScript

Notre projet contient deux fichiers JavaScript principaux dans le dossier `scripts/` :

```
scripts/
 main.js          # Interactions générales du site
 contact.js       # Validation du formulaire de contact
```

Ces fichiers sont chargés dans les pages HTML avec la balise `<script>` :

```
<script src="scripts/main.js"></script>
<script src="scripts/contact.js"></script>
```

## 1.2. Script principal : main.js

Le fichier `main.js` contient toutes les fonctionnalités JavaScript communes à l'ensemble du site. Nous avons utilisé une technique appelée IIFE (Immediately Invoked Function Expression) pour encapsuler notre code et éviter les conflits avec d'autres scripts.

### 1.2.1. Structure globale du script

Notre script principal est organisé comme suit :

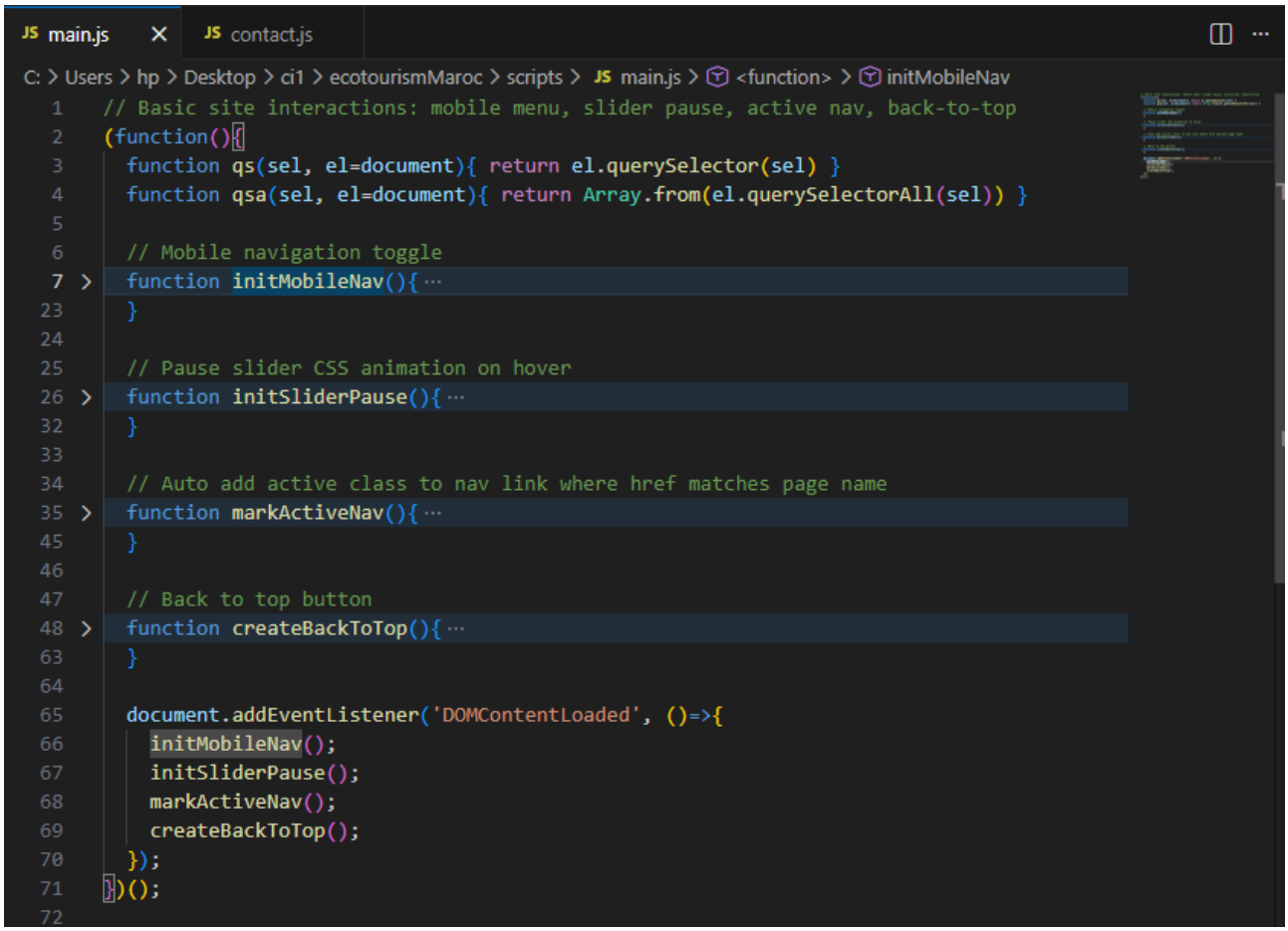


**Figure 1.1 –** main.js

### 1.2.2. Fonctions utilitaires

Nous avons créé deux fonctions utilitaires pour simplifier la sélection d'éléments HTML :

**Listing 1.1 –** Fonctions utilitaires de sélection

```
function qs(sel, el=document){
  return el.querySelector(sel)
}

function qsa(sel, el=document){
  return Array.from(el.querySelectorAll(sel))
}
```

**Explication :**

- `qs` : Raccourci pour `querySelector` - sélectionne UN élément
- `qsa` : Raccourci pour `querySelectorAll` - sélectionne TOUS les éléments correspondants
- `el=document` : Paramètre par défaut, cherche dans tout le document si non spécifié
- `Array.from()` : Convertit la liste d'éléments en tableau pour faciliter les manipulations

Ces fonctions nous permettent d'écrire du code plus court et lisible.

## 1.3. Menu de navigation mobile

L'une des fonctionnalités principales de notre site est le menu mobile responsive. Sur les petits écrans, le menu de navigation se transforme en menu hamburger.

### 1.3.1. Principe de fonctionnement

Le menu mobile fonctionne selon le principe suivant :

1. Un bouton hamburger (trois barres) est visible sur mobile
2. Au clic sur ce bouton, le menu s'ouvre ou se ferme
3. L'utilisateur peut aussi fermer le menu en appuyant sur la touche Échap
4. L'attribut ARIA `aria-expanded` indique l'état ouvert/fermé pour l'accessibilité

### 1.3.2. Code du menu mobile

Voici le code complet que nous avons écrit :

```
function initMobileNav(){
// Selectionner le bouton toggle et le header
const toggle = qs('.nav-toggle');
const header = qs('header');
// Verifier que les elements existent
if(!toggle || !header) return;
// Ecouter les clics sur le bouton
toggle.addEventListener('click', ()=>{
  // Lire l'etat actuel du menu
  const expanded = toggle.getAttribute('aria-expanded') === 'true';
  // Inverser l'etat
  toggle.setAttribute('aria-expanded', String(!expanded));
  // Basculer la classe CSS qui affiche/cache le menu
  header.classList.toggle('nav-open');
});
// Fermer le menu avec la touche Echap
document.addEventListener('keydown', (e)=>{
  if(e.key === 'Escape' && header.classList.contains('nav-open')){
    header.classList.remove('nav-open');
    toggle.setAttribute('aria-expanded','false');
  }
});
}
```

**Figure 1.2 –** main.js / initMobileNav()

### 1.3.3. Explication détaillée

**Étape 1 : Sélection des éléments**

```
const toggle = qs('.nav-toggle');
const header = qs('header');
```

On récupère le bouton hamburger (classe `.nav-toggle`) et l'élément `<header>`.

**Étape 2 : Vérification de l'existence**

```
if(!toggle || !header) return;
```

Si un élément n'existe pas sur la page, on arrête la fonction pour éviter les erreurs.

**Étape 3 : Gestion du clic**

```
toggle.addEventListener('click', ()=>{ /* ... */ });
```

On écoute les clics sur le bouton hamburger.

**Étape 4 : Lecture de l'état actuel**

```
const expanded = toggle.getAttribute('aria-expanded') === 'true';
```

On vérifie si le menu est déjà ouvert en lisant l'attribut `aria-expanded`.

**Étape 5 : Inversion de l'état**

```
toggle.setAttribute('aria-expanded', String(!expanded));
header.classList.toggle('nav-open');
```

On inverse l'état : si ouvert, on ferme ; si fermé, on ouvre.

**Étape 6 : Fermeture au clavier**

```
document.addEventListener('keydown', (e)=>{
  if(e.key === 'Escape' && header.classList.contains('nav-open')){
    // Fermer le menu
  }
});
```

On permet à l'utilisateur de fermer le menu en appuyant sur Échap.

### 1.3.4. Le CSS correspondant

Le JavaScript ajoute/retire simplement la classe `nav-open`. C'est le CSS qui gère l'apparence :

```
/* Menu cache par defaut sur mobile */
header nav {
  display: none;
}

/* Menu visible quand la classe nav-open est presente */
header.nav-open nav {
  display: block;
}
```

## 1.4. Animation du slider d'images

Notre site possède un slider (carrousel) d'images qui défile automatiquement sur la page d'accueil. Nous avons ajouté une fonctionnalité pour mettre en pause l'animation quand l'utilisateur survole le slider avec sa souris.

### 1.4.1. Principe du slider

Le slider fonctionne ainsi :

— Les images défilent automatiquement grâce à une animation CSS

— Quand l'utilisateur survole le slider, l'animation se met en pause

— Quand l'utilisateur enlève sa souris, l'animation reprend

### 1.4.2. Code de la pause du slider

```
// Pause slider CSS animation on hover
function initSliderPause(){
// Selectionner le conteneur du slider et les slides
const slider = qs('.slider');
const slide = qs('.slide');
// Verifier que les elements existent
if(!slider || !slide) return;
// Quand la souris entre dans le slider
slider.addEventListener('mouseenter', ()=> {
  slide.classList.add('paused');
});
// Quand la souris sort du slider
slider.addEventListener('mouseleave', ()=> {
  slide.classList.remove('paused');
});
}
```

**Figure 1.3 –** main.js / initSliderPause()

### 1.4.3. Explication du code

1. **Sélection** : On récupère le conteneur `.slider` et l'élément `.slide`
2. **mouseenter** : Événement déclenché quand la souris entre dans la zone du slider
   — On ajoute la classe `paused` qui arrête l'animation CSS
3. **mouseleave** : Événement déclenché quand la souris sort de la zone
   — On retire la classe `paused`, l'animation reprend

### 1.4.4. Animation CSS correspondante

L'animation est définie en CSS :

```
.slide {
  animation: slideAnimation 15s infinite;
}

.slide.paused {
  animation-play-state: paused;
}

@keyframes slideAnimation {
  0%, 100% { transform: translateX(0); }
  33% { transform: translateX(-100%); }
  66% { transform: translateX(-200%); }
}
```

## 1.5. Marquage du lien actif dans la navigation

Pour améliorer l'expérience utilisateur, nous avons créé une fonction qui met automatiquement en évidence le lien de navigation correspondant à la page actuelle.

### 1.5.1. Objectif

Si l'utilisateur est sur la page `destinations.html`, le lien "Destinations" dans le menu doit avoir un style différent (couleur, soulignement, etc.) pour indiquer qu'il s'agit de la page actuelle.

### 1.5.2. Code de marquage actif

```
// Auto add active class to nav link where href matches page name
function markActiveNav(){
  const path = location.pathname.split('/').pop() || 'index.html';
  qsa('nav.main-nav a').forEach(a=>{
    const href = a.getAttribute('href');
    if(!href) return;
    if(path === href || (href.endsWith('index.html') && path === '')){
      a.classList.add('active');
    }
  })
}
```

**Figure 1.4 –** main.js / markActiveNav()

### 1.5.3. Explication pas à pas

**1. Récupération du nom de la page :**

```
const path = location.pathname.split('/').pop() || 'index.html';
```

- `location.pathname` : Donne le chemin de l'URL actuelle
- `split('/')` : Découpe le chemin en morceaux
- `.pop()` : Prend le dernier morceau (le nom du fichier)
- `|| 'index.html'` : Si vide, utilise 'index.html' par défaut

Exemple : Si l'URL est `https://site.com/pages/contact.html`, `path` vaudra `contact.html`

**2. Parcours des liens :**

```
qsa('nav.main-nav a').forEach(a=>{ /* ... */ })
```

On sélectionne tous les liens (<a>) dans la navigation et on les parcourt un par un.

**3. Comparaison et marquage :**

```
if(path === href ||
   (href.endsWith('index.html') && path === '')){
  a.classList.add('active');
}
```

Si le `href` du lien correspond à la page actuelle, on ajoute la classe `active`.

## 1.6. Bouton de retour en haut de page

Pour améliorer la navigation sur les pages longues, nous avons créé un bouton "Retour en haut" qui apparaît automatiquement quand l'utilisateur descend dans la page.

### 1.6.1. Fonctionnement

— Le bouton est caché par défaut

— Il apparaît quand l'utilisateur a scrollé plus de 200 pixels vers le bas

— Au clic, la page remonte en haut avec une animation fluide

— Le bouton disparaît quand l'utilisateur est en haut de page

### 1.6.2. Code du bouton retour en haut

```
// Back to top button
function createBackToTop(){
  const btn = document.createElement('button');
  btn.className = 'back-to-top';
  btn.title = 'Remonter en haut';
  btn.innerText = '↑';
  btn.setAttribute('aria-label','Retour en haut');
  btn.style.display = 'none';
  document.body.appendChild(btn);

  btn.addEventListener('click', ()=> window.scrollTo({top:0,behavior:'smooth'}));

  window.addEventListener('scroll', ()=>{
    if(window.scrollY > 200) btn.style.display = 'block';
    else btn.style.display = 'none';
  })
}
```

**Figure 1.5 –** main.js / createBackToTop()

### 1.6.3. Explication détaillée

**Création du bouton :**

```
const btn = document.createElement('button');
btn.className = 'back-to-top';
btn.innerText = '\u2191';
```

On crée un nouvel élément `<button>` en JavaScript, on lui donne une classe CSS et on ajoute une flèche ↑.

**Ajout à la page :**

```
document.body.appendChild(btn);
```

On insère le bouton à la fin du `<body>`.

**Action au clic :**

```
btn.addEventListener('click', ()=> {
  window.scrollTo({top:0, behavior:'smooth'});
});
```

Quand on clique, `window.scrollTo()` fait remonter la page en haut (`top:0`) avec une animation fluide (`behavior:'smooth'`).

**Affichage conditionnel :**

```
window.addEventListener('scroll', ()=>{
  if(window.scrollY > 200) {
    btn.style.display = 'block';
  } else {
    btn.style.display = 'none';
  }
})
```

On écoute l'événement `scroll`. Si `window.scrollY` (position de scroll verticale) dépasse 200 pixels, on affiche le bouton, sinon on le cache.

## 1.7. Initialisation au chargement de la page

Toutes nos fonctions sont appelées quand la page est complètement chargée :

```
document.addEventListener('DOMContentLoaded', ()=>{
  initMobileNav();
  initSliderPause();
  markActiveNav();
  createBackToTop();
});
```

FIGURE 1.6

**Pourquoi DOMContentLoaded ?**

L'événement `DOMContentLoaded` est déclenché quand tout le HTML est chargé et que le DOM (Document Object Model) est prêt à être manipulé. C'est important car si on essaie de sélectionner des éléments avant qu'ils existent, le code ne fonctionnera pas.

## 1.8. Validation du formulaire de contact

Le deuxième fichier JavaScript `contact.js` gère entièrement la validation du formulaire de contact. C'est le script le plus complexe de notre projet car il vérifie de nombreux champs différents.

### 1.8.1. Structure du formulaire HTML

Notre formulaire de contact contient les champs suivants :

```
<form id="contactForm">
  <input id="fullname" type="text" placeholder="Nom complet">
  <input id="age" type="number" placeholder="Age">
  <input id="email" type="email" placeholder="Email">
  <input id="password" type="password" placeholder="Mot de passe">
  <input id="confirmPassword" type="password"
         placeholder="Confirmer le mot de passe">
  <input id="fileInput" type="file">
  <textarea id="message" maxlength="500"
            placeholder="Votre message"></textarea>
  <button type="submit">Envoyer</button>
</form>
```

### 1.8.2. Initialisation des variables

Au début du script, on sélectionne tous les éléments du formulaire :

```
document.addEventListener('DOMContentLoaded', () => {
  const form = document.getElementById('contactForm');
  const fullname = document.getElementById('fullname');
  const age = document.getElementById('age');
  const email = document.getElementById('email');
  const password = document.getElementById('password');
  const confirmPassword = document.getElementById('confirmPassword');
  const fileInput = document.getElementById('fileInput');
  const message = document.getElementById('message');
  const messageCounter = document.getElementById('messageCounter');
  const filePreview = document.getElementById('filePreview');
  const formMessage = document.getElementById('formMessage');
```

FIGURE 1.7

**Explication :**

— On utilise `getElementById()` pour récupérer chaque champ par son ID

— `allowedExt` : tableau des extensions de fichiers autorisées

— Tout le code est dans un `DOMContentLoaded` pour s'assurer que les éléments existent

### 1.8.3. Fonctions utilitaires d'affichage des erreurs

Nous avons créé des fonctions pour afficher et effacer les messages d'erreur :

```
function setError(el, msg) {
  const container = el.closest('label') || el.parentElement;
  const error = container && container.querySelector('.error-message');
  if (error) error.textContent = msg;
}
function clearError(el) {
  const container = el.closest('label') || el.parentElement;
  const error = container && container.querySelector('.error-message');
  if (error) error.textContent = '';
}
function setStatus(text, type='success'){
  formMessage.textContent = text;
  formMessage.classList.remove('success','error');
  formMessage.classList.add(type);
}
function clearStatus(){
  formMessage.textContent = ''; formMessage.classList.remove('success','error');
 }
```

FIGURE 1.8 – contact.js / fonctions d'erreurs

**Comment ça marche :**

— `setError()` : Affiche un message d'erreur sous un champ

— `clearError()` : Efface le message d'erreur

— `setStatus()` : Affiche un message global (succès ou erreur)

— `clearStatus()` : Efface le message global

### 1.8.4. Effacement automatique des erreurs

Pour améliorer l'expérience utilisateur, on efface les erreurs dès que l'utilisateur commence à corriger :

```
// Clear a field's error as user types/corrects
[fullname, age, email, password, confirmPassword, fileInput, message].forEach(el => {
  if (!el) return;
  el.addEventListener('input', () => { clearError(el); clearStatus(); });
});
```

**Figure 1.9 –** contact.js / fonctions d'erreurs

On parcourt tous les champs et on ajoute un écouteur d'événement `input` qui efface l'erreur dès que l'utilisateur tape quelque chose.

### 1.8.5. Compteur de caractères pour le message

Le champ message est limité à 500 caractères. Nous affichons un compteur en temps réel :

```
const MAX_MESSAGE = 500;
function updateMessageCounter(){
  if (!message || !messageCounter) return;
  const len = message.value.length;
  messageCounter.textContent = `${len} / ${MAX_MESSAGE}`;
  if (len > MAX_MESSAGE) {
    messageCounter.classList.add('warn');
    setError(message, `Le message ne doit pas dépasser ${MAX_MESSAGE} caractères`);
  } else if (len > (MAX_MESSAGE - 50)) {
    messageCounter.classList.add('warn');
    clearError(message);
  } else {
    messageCounter.classList.remove('warn');
    clearError(message);
  }
}
if (message){
  // initialize counter
  updateMessageCounter();
  message.addEventListener('input', () => {
    if (message.value.length > MAX_MESSAGE) message.value = message.value.slice(0, MAX_MESSAGE);
    updateMessageCounter();
  });
}
```

**Figure 1.10 –** contact.js / Compteur de caracteres

**Fonctionnement :**

1. On compte le nombre de caractères avec `message.value.length`
2. On affiche "X / 500"
3. Si on dépasse 500, on affiche un avertissement
4. Si on approche de 500 (entre 450 et 500), on ajoute une classe `warn` (couleur orange par exemple)
5. Si on dépasse vraiment, on coupe le texte avec `slice(0, 500)`

### 1.8.6. Prévisualisation et validation du fichier

Quand l'utilisateur choisit un fichier, on vérifie qu'il est valide et on affiche un aperçu :

```
fileInput && fileInput.addEventListener('change', () => {
  filePreview.innerHTML = '';
  clearError(fileInput);
  const f = fileInput.files && fileInput.files[0];
  if (!f) return;
  const name = f.name || '';
  const ext = name.split('.').pop().toLowerCase();
  if (!allowedExt.includes(ext)) {
    setError(fileInput, 'Type de fichier non autorisé. Seuls .pdf, .jpg, .jpeg sont acceptés');
    fileInput.value = '';
    return;
  }
  if (f.size > 5 * 1024 * 1024) {
    setError(fileInput, 'Fichier trop volumineux (max 5MB).');
    fileInput.value = '';
    return;
  }
  if (ext === 'jpg' || ext === 'jpeg'){
    const img = document.createElement('img');
    img.alt = name;
    filePreview.appendChild(img);
    const reader = new FileReader();
    reader.onload = e => { img.src = e.target.result; };
    reader.readAsDataURL(f);
    const meta = document.createElement('div'); meta.className = 'meta'; meta.textContent = name;
    filePreview.appendChild(meta);
  } else if (ext === 'pdf'){
    const meta = document.createElement('div'); meta.className = 'meta'; meta.textContent = `PDF
    filePreview.appendChild(meta);
  }
});
```

**Figure 1.11 –** contact.js /Validation des fichiers

**Explication étape par étape :**

1. **Récupération du fichier** : `fileInput.files[0]` donne le premier fichier sélectionné
2. **Extraction de l'extension** :
   - `name.split('.')` découpe le nom par les points
   - `.pop()` prend la dernière partie (l'extension)
   - `.toLowerCase()` convertit en minuscules
3. **Vérification de l'extension** : On vérifie que l'extension est dans notre liste autorisée
4. **Vérification de la taille** : 5 MB = 5 × 1024 × 1024 octets
5. **Prévisualisation** :
   - Pour les images : on crée un `<img>` et on utilise `FileReader` pour charger l'image
   - Pour les PDF : on affiche juste le nom du fichier

### 1.8.7. Validation de l'email

Nous avons créé une fonction pour vérifier que l'email est bien formaté :

```
function isEmail(v){
  return /^[\w-.]+@([\w-]+\.)+[\w-]{2,}$/.test(v);
}
```

**Figure 1.12 –** contact.js/ Validation d'email

Cette fonction utilise une expression régulière (regex) pour vérifier le format :

— `[\w-.]+` : Au moins un caractère (lettre, chiffre, tiret, point)

— `@` : Le symbole arobase obligatoire

— `([\w-]+\.)+` : Nom de domaine avec au moins un point

— `[\w-]{2,}` : Extension de domaine (au moins 2 caractères)

Exemples valides : `user@example.com`, `prenom.nom@domaine.fr`
Exemples invalides : `user@example`, `@example.com`, `user.example.com`

### 1.8.8. Validation complète au moment de la soumission

Quand l'utilisateur soumet le formulaire, on vérifie tous les champs :

```
form && form.addEventListener('submit', (ev) => {
  ev.preventDefault();
  clearStatus();
  let valid = true;

  const fName = fullname.value.trim();
  if (fName.length < 2) { setError(fullname, 'Nom trop court'); valid = false; }

  const a = parseInt(age.value, 10);
  if (!a || a < 18) { setError(age, 'Vous devez être âgé·e de 18 ans ou plus.'); valid = false; }
  const mail = email.value.trim();
  if (!isEmail(mail)) { setError(email, 'Veuillez entrer une adresse email valide'); valid = false; }
  const pass = password.value;
  if (pass.length < 6) { setError(password, 'Le mot de passe doit contenir au moins 6 caractères');
    valid = false; }
  const confirm = confirmPassword.value;
  if (confirm !== pass) { setError(confirmPassword, 'Les mots de passe ne correspondent pas');
    valid = false; }
  const f = fileInput.files && fileInput.files[0];
  if (f){
    const ext = (f.name.split('.').pop() || '').toLowerCase();
    if (!allowedExt.includes(ext)) { setError(fileInput, 'Format non autorisé'); valid = false; }
    if (f.size > 5 * 1024 * 1024){ setError(fileInput, 'Fichier trop volumineux (max 5MB)'); valid = fal
  }
  if (!valid){ setStatus('Le formulaire contient des erreurs. Corrigez les champs indiqués.', 'error');
  setStatus('Merci — votre message a été envoyé (simulation). Nous vous répondrons bientôt.', 'success')

  form.reset();
  filePreview.innerHTML = '';
  setTimeout(() => { clearStatus(); }, 5500);
```

**Figure 1.13 –** contact.js/Validation Du Formulaire

**Logique de validation :**

1. `ev.preventDefault()` : Empêche l'envoi automatique du formulaire
2. Variable `valid` : On part du principe que tout est valide (`true`), et on passe à `false` dès qu'on trouve une erreur

3. **Validation du nom** : Au moins 2 caractères après suppression des espaces (`trim()`)
4. **Validation de l'âge** :
   — `parseInt(age.value, 10)` : Convertit le texte en nombre entier
   — Vérifie que c'est un nombre valide et 18
5. **Validation de l'email** : Utilise notre fonction `isEmail()`
6. **Validation du mot de passe** : Au moins 6 caractères
7. **Confirmation du mot de passe** : Doit être identique au premier
8. **Validation du fichier** : Même vérifications que lors du changement
9. **Affichage du résultat** :
   — Si erreur : message d'erreur en rouge
   — Si succès : message de confirmation en vert
   — `form.reset()` : Vide tous les champs
   — `setTimeout()` : Efface le message après 5,5 secondes

### 1.8.9. Tests du menu mobile

1. Ouvrir le menu sur mobile → Fonctionne
2. Fermer avec la touche Échap → Fonctionne
3. Cliquer plusieurs fois sur le bouton → Ouvre/ferme correctement

### 1.8.10. Tests du formulaire

**Table 1.1 –** Tests de validation du formulaire

| Test | Résultat attendu | Résultat obtenu |
|---|---|---|
| Nom vide | Message d'erreur | OK |
| Âge < 18 | Message d'erreur | OK |
| Email sans @ | Message d'erreur | OK |
| Mots de passe différents | Message d'erreur | OK |
| Fichier .exe | Message d'erreur | OK |
| Fichier > 5MB | Message d'erreur | OK |
| Tout valide | Message de succès | OK |

### 1.8.11. Outils de débogage utilisés

— **Console du navigateur** : Pour afficher les erreurs JavaScript
— **Inspecteur d'éléments** : Pour vérifier les classes CSS ajoutées/retirées
— **Onglet Network** : Pour voir si les scripts se chargent correctement
— **Tests sur différents navigateurs** : Chrome, Firefox ,Edje

## 1.9. Conclusion du chapitre

Dans ce chapitre, nous avons détaillé l'implémentation de JavaScript dans notre projet ÉcoTourisme Maroc. Nous avons créé deux scripts principaux :

— **main.js** : Gère les interactions générales (menu mobile, slider, navigation active, bouton retour en haut)

— **contact.js** : Gère la validation complète du formulaire de contact avec vérification en temps réel

Ces fonctionnalités JavaScript améliorent considérablement l'expérience utilisateur en rendant le site plus interactif, plus réactif et plus agréable à utiliser. Bien que nous soyons débutants, nous avons réussi à implémenter des fonctionnalités importantes en suivant les bonnes pratiques du développement web moderne.

Le chapitre suivant présentera le déploiement du site en ligne sur GitHub Pages.

# 2

# Déploiement en ligne de la plateforme

Le déploiement d'un site web consiste à le rendre accessible au public via Internet. Dans ce chapitre, nous expliquons comment nous avons mis en ligne le site ÉcoTourisme Maroc en utilisant GitHub Pages, une solution d'hébergement gratuite et simple pour les sites statiques.

## 2.1. Qu'est-ce que le déploiement ?

Le déploiement est l'étape finale du développement web qui permet de :

— Rendre le site accessible à tout le monde via une URL

— Passer de l'environnement de développement (ordinateur local) à la production (serveur en ligne)

— Permettre aux utilisateurs de visiter le site depuis n'importe où dans le monde

### 2.1.1. Différence entre développement local et production

**Table 2.1 –** Développement local vs Production

| **Développement local** | **Production (en ligne)** |
|---|---|
| Fichiers sur votre ordinateur | Fichiers sur un serveur |
| Accessible uniquement par vous | Accessible par tout le monde |
| URL : localhost ou file :/// | URL : https ://site.com |
| Modifications instantanées | Nécessite redéploiement |

## 2.2. Choix de GitHub Pages

Pour héberger notre site, nous avons choisi GitHub Pages, une solution gratuite proposée par GitHub.

### 2.2.1. Pourquoi GitHub Pages ?

**Avantages :**

— **Gratuit** : Hébergement illimité sans frais

— **Simple** : Déploiement automatique en quelques clics

— **HTTPS gratuit** : Certificat SSL automatique pour sécuriser le site

— **CDN intégré** : Le site se charge rapidement partout dans le monde
— **Pas de publicité** : Contrairement aux hébergeurs gratuits classiques
— **Intégration Git** : Mise à jour facile via Git

**Limitations :**

— Uniquement pour les sites statiques (HTML, CSS, JavaScript)
— Pas de base de données
— Pas de PHP ou autres langages serveur

Pour notre projet (site vitrine statique), GitHub Pages est parfaitement adapté.

### 2.2.2. Alternatives considérées

Nous avons comparé plusieurs solutions :

**Table 2.2 –** Comparaison des solutions d'hébergement

| Service | Prix | Complexité |
|---|---|---|
| GitHub Pages | Gratuit | Facile |
| Netlify | Gratuit | Facile |
| Vercel | Gratuit | Moyenne |
| Hostinger | 2-10€/mois | Difficile |

GitHub Pages a été retenu pour sa simplicité et son intégration native avec Git.

## 2.3. Étapes du déploiement

Le déploiement de notre site sur GitHub Pages s'est fait en plusieurs étapes simples.

### 2.3.1. Étape 1 : Création du dépôt GitHub

Nous avons créé un dépôt (repository) sur GitHub pour stocker notre code :

1. Connexion à `github.com`
2. Clic sur "New repository"
3. Nom du dépôt : `ecotourismMaroc`
4. Visibilité : Public (obligatoire pour GitHub Pages gratuit)
5. Création du dépôt

### 2.3.2. Étape 2 : Initialisation de Git en local

Sur notre ordinateur, dans le dossier du projet :

**Explication des commandes :**

— `git init` : Initialise un nouveau dépôt Git
— `git add .` : Ajoute tous les fichiers au suivi Git
— `git commit -m "..."` : Enregistre les modifications avec un message
— `git remote add origin ...` : Connecte le dépôt local à GitHub
— `git push` : Envoie le code sur GitHub

### 2.3.3. Étape 3 : Activation de GitHub Pages

Dans les paramètres du dépôt sur GitHub :

1. Aller dans `Settings` (paramètres du dépôt)
2. Cliquer sur `Pages` dans le menu latéral
3. Dans "Source", sélectionner :
   — Branch : `main`
   — Folder : `/ (root)`
4. Cliquer sur `Save`
5. Attendre 1-2 minutes

GitHub Pages génère automatiquement le site à l'adresse :

`https://mouad-arr.github.io/ecotourismMaroc/`

### 2.3.4. Étape 4 : Vérification du déploiement

Après quelques minutes, nous avons :

1. Ouvert l'URL du site dans le navigateur
2. Vérifié que toutes les pages s'affichent correctement
3. Testé les liens de navigation
4. Vérifié que les images se chargent
5. Testé le menu mobile
6. Vérifié le formulaire de contact

**Résultat :** Le site est en ligne et accessible publiquement !

## 2.4. Mise à jour du site

Un des avantages de GitHub Pages est la facilité de mise à jour. Chaque fois que nous modifions le code et que nous le poussons sur GitHub, le site se met à jour automatiquement.

### 2.4.1. Processus de mise à jour

**Listing 2.1 –** Mise a jour du site

```
# 1. Faire des modifications dans le code

# 2. Voir les fichiers modifies
git status

# 3. Ajouter les modifications
git add .

# 4. Commit avec un message descriptif
git commit -m "Ajout: nouvelle destination Oasis du Sud"

# 5. Envoyer sur GitHub
git push origin main

# 6. Attendre 1-2 minutes : le site se met a jour automatiquement
```

**Temps de déploiement :** Entre 30 secondes et 2 minutes après le push.

### 2.4.2. Exemple de mise à jour

Nous avons fait plusieurs mises à jour après le déploiement initial :

— Correction de fautes d'orthographe

— Ajout de nouvelles images

— Amélioration du formulaire de contact

— Optimisation du menu mobile

Chaque mise à jour a suivi le même processus simple : modifier → commit → push.

## 2.5. Sécurité : HTTPS automatique

GitHub Pages active automatiquement HTTPS (protocole sécurisé) pour notre site. C'est très important pour plusieurs raisons :

### 2.5.1. Qu'est-ce que HTTPS ?

HTTPS (HyperText Transfer Protocol Secure) est la version sécurisée du HTTP. Il utilise le chiffrement SSL/TLS pour protéger les données.

**Avantages de HTTPS :**

— **Sécurité** : Les données échangées sont chiffrées

— **Confiance** : Les navigateurs affichent un cadenas vert

— **SEO** : Google favorise les sites HTTPS dans les résultats de recherche

— **Moderne** : Standard actuel du web

### 2.5.2. Configuration automatique

Avec GitHub Pages :

1. Le certificat SSL est généré automatiquement
2. Il se renouvelle automatiquement
3. Aucune configuration manuelle nécessaire
4. C'est totalement gratuit

Il suffit de cocher "Enforce HTTPS" dans les paramètres GitHub Pages (déjà activé par défaut).

### 2.5.3. Vérification du code

Nous avons vérifié que notre code HTML et CSS est correct :

— **HTML** : Validation avec le W3C Validator (`validator.w3.org`)

— **CSS** : Validation avec le CSS Validator

— **JavaScript** : Vérification des erreurs dans la console du navigateur

Toutes les erreurs détectées ont été corrigées avant le déploiement final.

### 2.5.4. Tests responsive

Nous avons testé l'affichage du site sur différentes tailles d'écran :

— Mobile (320px - 480px)

— Tablette (768px - 1024px)

— Desktop (> 1200px)

Tous les tests étaient positifs : le site s'affiche correctement partout.

## 2.6. Conclusion du chapitre

Le déploiement de notre site ÉcoTourisme Maroc sur GitHub Pages s'est déroulé avec succès. Nous avons réussi à :

— Mettre le site en ligne gratuitement

— Obtenir une URL publique et un certificat HTTPS

— Établir un processus simple pour les mises à jour futures

— Optimiser le site pour de bonnes performances

Le site est maintenant accessible à l'adresse `https://mouad-arr.github.io/ecotourismMaroc/` et peut être consulté par n'importe qui dans le monde.

Cette expérience de déploiement nous a permis de comprendre :

— Le fonctionnement de Git et GitHub

— La différence entre développement local et production

— L'importance de l'optimisation pour le web

— Le processus complet de mise en ligne d'un site web

# Conclusion générale

Le projet ÉcoTourisme Maroc nous a permis de découvrir et de pratiquer les technologies fondamentales du développement web moderne : HTML5, CSS3 et JavaScript. À travers ce projet, nous avons créé un site web complet, du code initial au déploiement en ligne.

## Objectifs atteints

Nous avons réussi à créer un site web fonctionnel qui répond aux objectifs fixés :

- **Site responsive** : Le site s'adapte correctement aux mobiles, tablettes et ordinateurs
- **Navigation intuitive** : Menu clair avec des liens fonctionnels vers toutes les pages
- **Design attractif** : Interface moderne avec des couleurs cohérentes et des images de qualité
- **Interactivité JavaScript** : Menu mobile, slider, formulaire validé, bouton retour en haut
- **Déploiement réussi** : Site en ligne et accessible publiquement

## Compétences acquises

Ce projet nous a permis d'apprendre et de pratiquer de nombreuses compétences techniques :

### HTML

- Structure sémantique des pages
- Formulaires et validation
- Organisation du contenu
- Balises meta et SEO de base

### CSS

- Flexbox et CSS Grid pour les layouts
- Media Queries pour le responsive design
- Animations et transitions
- Variables CSS pour la cohérence
- Gestion des couleurs et typographie

### JavaScript

- Manipulation du DOM
- Gestion des événements
- Validation de formulaires
- Création de fonctionnalités interactives
- Bonnes pratiques de code

### Git et déploiement

— Versioning du code avec Git
— Utilisation de GitHub
— Déploiement avec GitHub Pages
— Workflow de mise à jour

## Difficultés rencontrées et solutions

En tant que débutants, nous avons rencontré plusieurs défis :

### Responsive Design

**Difficulté :** Adapter le site aux différentes tailles d'écran
**Solution :** Utilisation de Flexbox, CSS Grid et Media Queries après étude de tutoriels et exemples

### JavaScript

**Difficulté :** Comprendre la logique de programmation et la manipulation du DOM
**Solution :** Découpage en petites fonctions, tests fréquents dans la console, documentation MDN

### Git

**Difficulté :** Comprendre les concepts de commit, push, branches
**Solution :** Apprentissage progressif des commandes de base, aide du professeur

## Points forts du projet

— **Fonctionnalité complète** : Le site contient toutes les pages prévues et toutes les fonctionnalités fonctionnent
— **Design cohérent** : Charte graphique respectée sur toutes les pages
— **Accessibilité** : Navigation au clavier, attributs ARIA, messages d'erreur clairs
— **Performance** : Images optimisées, code validé
— **En ligne** : Site déployé et accessible publiquement

## Améliorations futures possibles

Si nous devions continuer à développer ce site, nous pourrions ajouter :

### Court terme

— Plus de destinations et d'activités
— Galerie photos avec lightbox
— Système de recherche de destinations
— Blog avec articles sur l'écotourisme
— Page FAQ (questions fréquentes)

### Moyen terme

— Backend avec base de données
— Système de réservation en ligne

— Comptes utilisateurs
— Système de commentaires et avis
— Newsletter

### Long terme

— Application mobile
— Carte interactive avec géolocalisation
— Système de recommandations personnalisées
— Partenariats avec éco-lodges
— Intégration de paiement en ligne

## Impact du projet

Au-delà de l'apprentissage technique, ce projet a un objectif de sensibilisation au tourisme durable au Maroc. Notre site vise à :

— Promouvoir les destinations naturelles marocaines
— Encourager des pratiques de voyage responsables
— Valoriser le patrimoine culturel et environnemental
— Soutenir les communautés locales

## Remerciements

Nous tenons à remercier :

— Le Professeur Qazdar Aimad pour son encadrement et ses conseils
— L'ENSA pour la formation en technologies web
— Nos camarades pour les échanges et l'entraide

## Conclusion finale

Ce projet a été une expérience d'apprentissage très enrichissante. Partir d'une page blanche et arriver à un site web complet et déployé en ligne nous a donné confiance en nos capacités de développement web.

Nous avons découvert que créer un site web demande de la rigueur, de la patience et de la créativité. Chaque problème rencontré nous a appris quelque chose de nouveau. Le résultat final, bien qu'il soit celui de débutants, est fonctionnel et nous en sommes fiers.

Cette expérience nous a donné envie de continuer à apprendre le développement web et à améliorer nos compétences. Le site ÉcoTourisme Maroc est maintenant en ligne à l'adresse :

`https://mouad-arr.github.io/ecotourismMaroc/`

C'est le début d'une aventure dans le monde du développement web, et nous sommes motivés pour continuer à apprendre et à créer.

*Mouad Ouchelh, Youssef Afella, Achraf Boulhem*
*Février 2025*