Usages des techniques numériques

Maitriser les technologies du web

Module 1 – Panorama des technologies du web

Introduction

- Le web est un écosystème complexe, constitué de technologies, de langages et de protocoles qui interagissent pour rendre possible la communication numérique, la diffusion de contenus et l'interactivité que nous connaissons aujourd'hui.

- Ce module vise à en présenter les **fondations techniques**, à comprendre la **logique des architectures web**, des **langages** et des **systèmes de gestion de contenus** pour dialoguer avec les équipes de développement, d'intégration ou de design.

- Avant de manipuler des outils ou d'administrer un CMS, il est essentiel de comprendre **ce qui se passe sous la surface** : comment un site web fonctionne, comment une page s'affiche, et comment les données circulent entre le navigateur, le serveur et la base de données.

- **Objectifs** :
  1. Obtenir une **culture générale** des technologies du web, des langages et de leurs rôles respectifs,
  2. Préparer le terrain à la **mise en œuvre concrète** d'un site dynamique (sous CMS).

Sommaire

1. **Architecture client–serveur** (qui gouverne le web) ;
2. **Langages du front-end** (la partie visible du site) ;
3. **Langages du back-end** (la logique et le traitement des données) ;
4. **Bases de données et langage SQL** (le stockage et la manipulation de l'information) ;
5. **Frameworks modernes** (qui automatisent la production de contenu) ;
6. **CMS** (qui automatisent la production de contenu) ;
7. **Hébergement, DNS et mise en ligne**.

1. Architecture client-serveur

- L'architecture client–serveur est le modèle fondateur du web.
- Elle décrit la manière dont les échanges s'organisent entre deux entités :
  le **client**, c'est-à-dire le dispositif qui formule une demande (un navigateur web, une application, un smartphone…),
  et le **serveur**, la machine distante qui stocke, traite et envoie les réponses.

1.1. Principe général

- Lorsqu'un utilisateur saisit une adresse dans son navigateur, plusieurs actions invisibles se déroulent en quelques fractions de seconde :

  1. Le navigateur (client) envoie une **requête HTTP** ou **HTTPS** au serveur correspondant à l'adresse demandée.
  2. Le serveur **reçoit la requête**, identifie la ressource (fichier HTML, image, script ou page générée dynamiquement) et prépare une réponse.
  3. Le serveur **renvoie la réponse** au client sous forme d'un document web interprétable par le navigateur.
  4. Le navigateur **affiche le contenu** et exécute éventuellement les scripts ou feuilles de style associés.

- Ce modèle d'échange **constitue la base de fonctionnement** de presque tous les services numériques (sites vitrines, plateformes de streaming, boutiques en ligne, etc.).

1. Architecture client-serveur

1.2. Les composants clés

Élément	Rôle
Client	Dispositif qui envoie la requête : navigateur (Chrome, Safari, Firefox, Edge), app mobile, ou logiciel connecté.
Serveur web	Héberge les fichiers du site et distribue les pages demandées (Apache, Nginx, LiteSpeed).
Serveur applicatif	Interprète le code côté serveur (PHP, Node.js, Python, etc.).
Base de données	Stocke les informations dynamiques (articles, utilisateurs, produits…).
Réseau et protocoles	HTTP/HTTPS, DNS, TCP/IP assurent la communication et la sécurité des échanges.

1. Architecture client-serveur

1.3. Schéma conceptuel simplifié

[Navigateur client]	⇄	[Serveur web]	⇄	[Serveur applicatif]	⇄	[Base de données]
HTML, CSS, JS		PHP, Python, Node.js				SQL, données

Le navigateur (client) affiche le contenu et exécute les scripts.
Le serveur web reçoit les requêtes, puis interagit avec un serveur applicatif ou une base de données pour générer la réponse.

1.4. Modèle client-serveur

Comprendre le modèle client–serveur permet :

- de **visualiser la circulation des données** ;
- d'**identifier les rôles des différentes couches** techniques ;
- et de **dialoguer efficacement** avec des développeurs, intégrateurs ou administrateurs système.

> Ce modèle est une structure invisible mais essentielle.

1. Architecture client-serveur

1.5. Les modèles d'architecture les plus utilisés aujourd'hui

Modèle	Principe	Usage typique	Exemples concrets
Architecture à 3 tiers (classique)	Séparation claire entre **front-end**, **back-end** et **base de données**. Le front-end s'affiche dans le navigateur, le back-end génère les pages et dialogue avec la base.	Modèle dominant dans la plupart des **sites web dynamiques** (WordPress, Prestashop, Laravel…).	WordPress, Symfony, Django
Architecture distribuée / microservices	Chaque fonction du système (paiement, authentification, recherche, etc.) est un **service autonome**, qui communique via des **API REST**	Utilisé pour les **grandes plateformes** nécessitant performance, modularité et scalabilité.	Netflix, Amazon, Uber
Serverless	Le code n'est pas hébergé en continu : il s'exécute **à la demande**, dans le cloud. Facturation à l'usage.	Très répandu pour les **applications modernes**, les **API légères** ou les **sites statiques dynamisés**.	AWS Lambda, Google Cloud Functions, Vercel

2. Langages du front-end

- Le **front-end** désigne la partie d'un site ou d'une application web que l'utilisateur voit, l'interface avec laquelle l'utilisateur interagit.
- Il est responsable de la présentation (les textes, les images, les formulaires), de la navigation (les menus) et de l'interactivité (les animations).
- Il s'agit de la **couche de présentation**, celle qui relie la logique technique à l'expérience utilisateur.

2. Langages du front-end

2.1. Les trois langages fondamentaux du front-end

Langage	Rôle	Exemple d'usage
HTML (*HyperText Markup Language*)	C'est le squelette du web. Il structure le contenu grâce à des balises : titres <h1>, paragraphes <p>, liens <a>, listes , formulaires <form>... HTML ne décrit pas l'apparence, mais la hiérarchie et le sens des informations (approche sémantique).	Définir la structure d'un article de blog ou la hiérarchie des titres d'une page.
CSS (*Cascading Style Sheets*)	C'est la peau du web. Les feuilles de style contrôlent la mise en forme : couleurs, typographies, espacements, animations, transitions. Elles permettent le **responsive design** (adaptation automatique de la page aux écrans de différentes tailles).	Mettre en forme un menu horizontal ou une grille d'images.
JavaScript	C'est le moteur du web côté client. Il rend la page dynamique et interactive : menus déroulants, formulaires en temps réel, notifications, appels API, etc. Aujourd'hui, JavaScript s'exécute dans tous les navigateurs modernes et communique avec le back-end via des requêtes asynchrones (AJAX, Fetch API).	Créer un carrousel d'images ou un bouton « charger plus d'articles ».

2. Langages du front-end

2.2. Compléments et outils modernes

- **Responsive design** : adaptation automatique des pages à la taille de l'écran (mobile, tablette, desktop).
  → Techniques : *media queries*, unités relatives (em, %, vh), et flexbox / grid.
- **Frameworks CSS** : outils de mise en page rapide (Bootstrap, Tailwind, Bulma).
  → Exemple : *Bootstrap* est utilisé dans des milliers de sites d'administration ou de start-up.
- **Préprocesseurs CSS** : Sass, Less — ajoutent des fonctions avancées (variables, mixins).
- **Bibliothèques JavaScript** : pour enrichir l'interactivité sans tout recoder (React, Vue, jQuery).

2.3. Évolution historique :

- **Années 1990** : HTML statique et CSS rudimentaire.
- **Années 2000** : apparition de JavaScript et du *DOM scripting*.
- **Années 2006** : essor du CSS complexe
- **Années 2010** : essor des frameworks (Angular, React, Vue).
- **Aujourd'hui** : front-end modulaire, réactif, interfacé avec des API.

2. Langages du front-end

2.4. Exemples de sites représentatifs

Langage / Technologie	Utilisation concrète
HTML + CSS	Pages statiques et documentaires : documentation GitHub, sites institutionnels.
HTML + CSS + JS	Sites interactifs : portfolios, boutiques en ligne, blogs dynamiques.
React (JS)	**Netflix**, **Instagram Web**, **Airbnb**, **Facebook**, **BBC**.
Vue.js	**GitLab**, **Xiaomi**, **Nintendo Europe**, **Upwork**, **Behance**.
Angular	**Google Workspace**, **Santander**, **Forbes**, **IBM Cloud**.

2. Langages du front-end

2.5. Bonnes pratiques du Front-end

- Code **accessible** : structuration logique, textes alternatifs, contraste suffisant.
- Code **performant** : limitation des scripts, compression d'images, chargement différé.
- Code **modulaire** : composants réutilisables et séparés (pattern : atomic design).
- **Tests et validation** : Lighthouse, WAVE, outils de validation W3C.

> À retenir

- Le front-end est le visage du web : il relie la technique à l'expérience utilisateur.
- Son évolution reflète l'histoire du web : du document statique (HTML 1.0) à l'application interactive connectée à des APIs (Web 3.0).

3. Frameworks front-end modernes : React, Vue, Angular, Svelte

- Les frameworks front-end modernes ont profondément transformé le développement web.
  Là où le JavaScript classique manipulait directement le DOM, ces outils offrent aujourd'hui une **structure modulaire**, des **composants réutilisables**, et une **interactivité fluide** entre les données et l'affichage.

3.1. Le principe des frameworks

- Un framework front-end est un **ensemble d'outils et de conventions** permettant de construire des interfaces riches et dynamiques.
  Leur promesse : réduire le code répétitif et simplifier la gestion de l'état d'une application.

- Plutôt que de manipuler le HTML manuellement, on crée des **composants** :
  des blocs autonomes qui gèrent leur contenu, leur style et leurs interactions.
  Ces composants s'actualisent automatiquement dès que les données changent (approche dite *reactive programming*).

3. Frameworks front-end modernes : React, Vue, Angular, Svelte

3.2. Les grands frameworks du web

Framework	Langage	Philosophie / Particularités	Exemples d'usage actuel (2025)
React	JavaScript / TypeScript	Bibliothèque développée par Meta. Utilise un *Virtual DOM* pour un rendu ultra rapide. Écosystème immense, interfaçable avec tout.	**Instagram**, **Netflix**, **Airbnb**, **Shopify**, **BBC**, **Uber Eats**, **Vercel**
Vue.js	JavaScript / TypeScript	Framework progressif : facile à apprendre, très flexible. Adapté aux projets modulaires et aux intégrations progressives.	**GitLab**, **Xiaomi**, **Nintendo**, **Behance**, **Alibaba**, **Statista**, **Upwork**
Angular	TypeScript	Framework complet soutenu par Google. Structure stricte, intégrant routage, injection de dépendance et tests. Idéal pour applications de grande ampleur.	**Google Workspace**, **Santander**, **Forbes**, **IBM Cloud**, **Microsoft 365 Web**, **Siemens**
Svelte / SvelteKit	JavaScript / TypeScript	Framework compilé : pas de runtime lourd, tout est transformé en JS pur au moment du build. Performances exceptionnelles.	**Apple Music Web**, **The New York Times**, **Square**, **Bloomberg**, **GoDaddy**, **Airtable dashboards**

3. Frameworks front-end modernes : React, Vue, Angular, Svelte

3.3. Évolutions et tendances 2025

- Montée du **TypeScript** : 90 % des nouveaux projets React et Angular l'utilisent.
- Essor des **meta-frameworks** (Next.js, Nuxt, Astro, Remix).
- Croissance des **micro-frontends** : architecture modulaire pour grands projets.

> À retenir

- Les frameworks front-end ont fait passer le web :
  - du **document statique** au **système applicatif dynamique**,
  - du **code HTML dispersé** au **composant réutilisable**,
  - du **chargement complet de page** au **rendu instantané des données**.

- Ils sont aujourd'hui la **colonne visible du web dynamique**, et leur compréhension est essentielle pour piloter ou concevoir un projet numérique contemporain.

4. Langages du back-end

- Le **back-end** est la partie invisible du web : c'est la couche qui traite les données, applique les règles de logique métier et dialogue avec les bases de données.

- Il représente la logique, la structure et la mémoire du web.
  C'est lui qui **gère les données, la sécurité, les accès, les calculs, et la communication entre le navigateur (client) et la base de données.**

- Quand vous naviguez sur un site, **chaque page dynamique affichée** (ex. un article, un profil utilisateur, une page produit) est le résultat du travail du back-end :

  une requête → un traitement → une réponse.

4. Langages du back-end

4.1. Rôles du back-end

a. Traiter les requêtes utilisateurs
→ Quand un formulaire est soumis ou qu'une page dynamique est demandée.

b. Accéder aux bases de données
→ Lire, écrire, mettre à jour ou supprimer des informations (opérations CRUD).

c. Générer des pages dynamiques
→ Injecter des données dans des modèles HTML pour créer la page envoyée au navigateur.

d. Gérer la sécurité et l'authentification
→ Sessions, mots de passe, rôles utilisateurs, autorisations d'accès.

e. Fournir des APIs
→ Servir de point d'accès pour d'autres applications (mobile, front React, etc.).

4. Langages du back-end

4.2. Les principaux langages web et leurs usages actuels

Langage	Particularités	Exemples d'usage réel
PHP	Simple, répandu, parfait pour la génération de contenu dynamique.	WordPress.com, Doctolib, OVH, Mailchimp, Wikipédia
Python	Lisible, modulaire, utilisé à la fois pour le web, la data et l'IA.	Instagram, Spotify, Dropbox, Reddit, Stripe
JavaScript / Node.js	Permet d'utiliser JS côté serveur : un langage unifié front/back.	Netflix, Trello, Uber, PayPal, Discord
C#	Performant, stable, utilisé surtout en entreprise et pour les intranets.	Stack Overflow, UPS, GoDaddy, Booking.com
Ruby	Syntaxe claire, orienté "convention plutôt que configuration".	Shopify, Basecamp, GitHub, Fiverr
Java	Robuste, multi-plateforme, idéal pour les architectures complexes et les microservices.	Amazon, LinkedIn, Twitter (back-end), Uber
Go (Golang)	Compilé, rapide, efficace pour les services cloud et microservices.	Cloudflare, Docker, Kubernetes, Twitch
Rust	Sécurisé et rapide, conçu pour la performance et la fiabilité.	AWS, Figma backend, Dropbox, Cloudflare
Elixir	Excellent pour les systèmes temps réel et la haute concurrence.	Discord, Pinterest (chat), Change.org

4. Langages du back-end

4.3. Les frameworks web majeurs par langage

Frameworks / outils les plus utilisés	Langage	Particularité du framework
Laravel, **Symfony**	**PHP**	MVC, simplicité, écosystème mature
Django, **Flask**, **FastAPI**	**Python**	Rapide à déployer, adapté à la data et à l'IA
Express, **NestJS**, **Next.js**, **Nuxt.js**	**JavaScript / TypeScript**	Unifie front/back, idéal pour SPAs et APIs
ASP.NET Core, **Blazor**	**C#**	Intégré à l'écosystème Microsoft
Ruby on Rails	**Ruby**	Très productif, approche Convention over Configuration
Spring Boot, **Micronaut**, **Quarkus**	**Java**	Fiable, très utilisé en entreprise

4. Langages du back-end

4.4. Frameworks vs CMS

Aspect	Nature	Objectif	Public visé	Souplesse
Framework	Outil de développement	Créer une application sur mesure	Développeurs	Très flexible, tout est possible, mais à coder
CMS	Application déjà construite	Gérer et publier du contenu	Utilisateurs / administrateurs	Moins flexible, mais rapide à mettre en œuvre

4. Langages du back-end

4.5. Tendances du back-end

- **Full JavaScript / TypeScript** : un même langage pour front et back.
- **Architecture API-first** : le back-end sert avant tout à fournir des données (REST ou GraphQL).
- **Micro-services et conteneurs** : découpage du code pour plus d'agilité.
- **Sécurité** et **authentification OAuth2 / JWT** : devenues centrales.
- **PHP** reste dominant en volume de sites web (grâce à WordPress et Laravel).

4. Langages du back-end

4.6. Exemple d'application complète

Lorsqu'un utilisateur publie un post sur **Reddit** :

1. Le front-end (React) envoie la requête via une API.
2. Le back-end (Python/Django + PostgreSQL) enregistre le post et renvoie un identifiant.
3. L'API Redis met à jour le fil en temps réel.
4. Le CDN (Cloudflare) sert le contenu mis en cache pour accélérer l'affichage.

> À retenir

- Le back-end est le **moteur logique** du web.
- Il orchestre les échanges, sécurise les données et sert d'interface entre le contenu, les utilisateurs et les services externes.

5. Bases de données et langage SQL

- Les bases de données sont le **cœur informationnel** d'un site web.

- Elles permettent de stocker, organiser et interroger efficacement des volumes de données structurées.
  Sans elles, aucun site dynamique — blog, e-commerce, réseau social — ne pourrait fonctionner.

5.1. Les rôles fondamentaux d'une base de données

a. **Le stockage structuré** de l'information : articles, comptes utilisateurs, commentaires, produits…
b. **La mise à jour et la suppression** des données selon les actions de l'utilisateur.
c. **La consultation rapide** de données précises grâce à des requêtes ciblées.
d. **La sécurité et la cohérence** des informations, grâce à des mécanismes de transactions et de permissions.

> Le lien entre le **serveur** et la base de données est essentiel :
le back-end envoie des requêtes pour interagir avec la bdd, puis restitue les résultats au front-end sous forme de pages HTML ou de données JSON.

5. Bases de données et langage SQL

5.2. Types de bases de données

5.2.1. Les bases relationnelles (SQL)

- Elles reposent sur un modèle en **tables** reliées entre elles par des clés (primaire et étrangère). Chaque table contient des colonnes (attributs) et des lignes (enregistrements).

Exemple de tables	Relation
utilisateurs (id, nom, email, rôle)	articles (id, titre, contenu, auteur_id)

La clé auteur_id relie un article à son auteur via l'ID utilisateur.

- Les SGBD (*Systèmes de Gestion de Bases de Données*) relationnels les plus utilisés sont :
  - **MySQL / MariaDB** (majoritaires sur le web)
  - **PostgreSQL** (open-source, très robuste)
  - **SQLite** (léger, sans serveur, pour prototypes)
  - **SQL Server** (environnement Microsoft)
  - **Oracle Database** (solutions d'entreprise)

5. Bases de données et langage SQL

5.2. Types de bases de données

5.2.2. Les bases non-relationnelles (NoSQL)

- Elles sont apparues pour gérer des données volumineuses, non structurées ou évolutives : documents, objets JSON, graphes, etc.

Type de NoSQL	Structure	Exemples d'outils	Usages typiques
Documentaire	Documents JSON	MongoDB, CouchDB, Firestore	Applications web, API REST
Clé–valeur	Données indexées par clé unique	Redis, DynamoDB	Caches, sessions utilisateur
Colonnaire	Données organisées par colonne	Cassandra, Bigtable	Analyse big data
Graphes	Relations orientées entre nœuds	Neo4j	Réseaux sociaux, recommandations

5. Bases de données et langage SQL

5.3. Le langage SQL (Structured Query Language)

- SQL est un langage de manipulation de données standardisé.
  Il permet d'effectuer les opérations dites **CRUD** :
  - **Create** : insérer des données
  - **Read** : lire des données
  - **Update** : modifier des données
  - **Delete** : supprimer des données

- **Exemples de requêtes :**

```
-- Sélection de tous les articles publiés
SELECT titre, auteur FROM articles WHERE statut = 'publié';

-- Ajout d'un nouvel utilisateur
INSERT INTO utilisateurs (nom, email) VALUES ('Alice Dupont', 'alice@example.com');

-- Mise à jour d'un mot de passe
UPDATE utilisateurs SET mot_de_passe = 'nouveauhash' WHERE id = 5;

-- Suppression d'un commentaire
DELETE FROM commentaires WHERE id = 12;
```

5. Bases de données et langage SQL

5.4. **Exemple de cycle de données** (site d'actualités)

a. L'utilisateur soumet un article via un formulaire web.
b. Le **back-end** (PHP) reçoit les données et exécute une requête INSERT INTO articles... pour les stocker.
c. Lorsqu'un visiteur consulte la page, le serveur exécute une requête SELECT pour afficher les derniers articles publiés.
d. Si un administrateur supprime un contenu, la requête DELETE le retire de la base.

> Ce cycle est typique de tous les systèmes dynamiques : il illustre le **lien constant entre l'interface visible et la structure de stockage invisible**.

5. Bases de données et langage SQL

5.5. Évolutions récentes

- Les bases **hybrides SQL/NoSQL**
- L'émergence des **bases distribuées** (CockroachDB, PlanetScale).
- L'usage croissant du **cloud database management** (AWS RDS, Google Cloud SQL).
- La montée en puissance de **SQLite** pour les prototypes et applications locales.

> À retenir

- La base de données est la **mémoire** du site web.
- Elle est invisible pour l'utilisateur, mais centrale pour la logique applicative.
- Maîtriser son fonctionnement conceptuel permet de comprendre comment un CMS ou une application web manipule, affiche et protège les données.

6. CMS (Content Management System)

Gérer, structurer et publier du contenu sans coder

6.1. Définition et principe général

- Un **CMS** (*Content Management System*) est un système de gestion de contenu permettant de **créer, éditer, organiser et publier** un site web sans écrire de code.
  Il repose sur une interface d'administration graphique qui communique avec une base de données et des modèles de pages (templates) pour produire un site complet.

- **Le principe :**
  Un utilisateur interagit avec une interface → le CMS stocke le contenu en base → le moteur du CMS génère dynamiquement la page web à afficher.

> le CMS automatise la gestion du contenu et la mise en forme sans passer par la programmation.

6. CMS (Content Management Systems)

6.2. Fonctionnement global d'un CMS

Élément	Rôle
Interface d'administration	Zone d'accès réservée (souvent /wp-admin, /drupal, /admin), utilisée pour créer et gérer les contenus.
Base de données	Stocke tous les contenus : textes, utilisateurs, médias, paramètres, etc.
Thèmes / Templates	Définissent l'apparence du site (structure HTML + styles CSS).
Plugins / Modules	Extensions ajoutant des fonctionnalités : SEO, formulaires, sécurité, e-commerce…
Système de rôles	Définit les droits des utilisateurs (administrateur, éditeur, auteur, contributeur, abonné…).

> Ce découpage rend la maintenance, la mise à jour et la collaboration beaucoup plus simples que sur un site codé à la main.

6. CMS (Content Management Systems)

6.3. Les CMS les plus utilisés en 2025

CMS	Type	Particularités	Exemples d'usage réel (2025)
WordPress	CMS généraliste (PHP/MySQL)	Simple, extensible, très populaire (**43 % du web**).	**BBC America**, **The Walt Disney Company**, **WhiteHouse.gov**, **Rolling Stone**, **Université de Lyon**
Drupal	CMS modulaire (PHP/PostgreSQL)	Robuste, orienté sites institutionnels complexes.	**ONU**, **NASA**, **Harvard**, **France Télévisions**, **Le CNRS**
Joomla!	CMS communautaire (PHP/MySQL)	Polyvalent, moins populaire mais très stable.	**MTV Greece**, **IKEA Catalog**, **Gouvernement grec**, **eBay (communautés)**
Ghost	CMS orienté blogging (Node.js/SQLite)	Léger, rapide, interface minimaliste.	**OpenAI Blog**, **Mozilla Foundation**, **Apple Developer News**
Strapi	CMS headless (Node.js + API REST/GraphQL)	Fournit uniquement le contenu via API, sans interface front-end.	**Toyota**, **Deliveroo**, **IBM Cloud**, **Virgin Media**, **Allianz**
Webflow / Wix / Squarespace	CMS SaaS hébergés	Environnements graphiques complets sans code.	**Dropbox Design**, **Zendesk Blog**, **Kickstarter**, **Headspace**

6. CMS (Content Management Systems)

6.4. Pourquoi utiliser un CMS ?

- **Accessibilité** : gestion possible sans compétence technique.
- **Gain de temps** : création rapide d'un site fonctionnel.
- **Personnalisation** : milliers de thèmes et plugins disponibles.
- **Collaboration** : rôles utilisateurs et workflow éditorial.
- **Évolutivité** : ajout progressif de fonctionnalités.
- **Standardisation** des structures de données (utile pour les migrations ou API).
- **Interopérabilité** avec les outils modernes (e-commerce, SEO, analytics, CRM, newsletter…).

6. CMS (Content Management Systems)

6.5. Les limites d'un CMS

- **Performance** parfois inférieure à un site statique (surtout sur WordPress mal configuré).
- **Sécurité** : surface d'attaque plus large (plugins obsolètes, mots de passe faibles).
- **Dépendance** à des extensions tierces.
- **Maintenance** nécessaire (sauvegardes, mises à jour, nettoyage de base).
- **Rigidité** : structure imposée par les modèles de contenu.

> À retenir

- Le CMS est la **colonne éditoriale** du web.
- Il a démocratisé la publication en ligne tout en s'adaptant à des besoins professionnels avancés.
- Sa compréhension est indispensable pour piloter des projets numériques ou travailler entre contenu et technique.

7. Hébergement, DNS et mise en ligne

7.1. De la création à la mise en ligne

- Une fois le site développé ou configuré sur un CMS, il doit être **hébergé** sur un serveur accessible en ligne.
- L'hébergement web permet de stocker les fichiers du site (HTML, CSS, JS, médias) et la base de données sur un ordinateur connecté en permanence à Internet.
- Le site est alors accessible via un **nom de domaine** unique (ex. : monsite.fr).

7. Hébergement, DNS et mise en ligne

7.2. Les principaux types d'hébergement

Type d'hébergement	Description	Avantages	Inconvénients
Mutualisé	Plusieurs sites partagent les mêmes ressources serveur.	Peu coûteux, facile à gérer.	Moins performant, configuration limitée.
VPS (Virtual Private Server)	Serveur virtuel dédié à un seul utilisateur.	Plus flexible, bon rapport puissance/prix.	Configuration technique requise.
Serveur dédié	Machine physique entièrement réservée à un seul site ou client.	Puissance maximale, contrôle total.	Coût élevé, gestion complexe.
Cloud / Elastic hosting	Ressources distribuées et ajustables selon la charge.	Évolutif, haute disponibilité.	Coût variable, dépendance fournisseur.

7. Hébergement, DNS et mise en ligne

7.3. Hébergeurs courants et infrastructures

- **O2switch** (France) – offre tout-en-un mutualisée, idéale pour CMS (WordPress, Joomla).
- **OVHcloud** – leader européen, VPS et cloud public.
- **Infomaniak** – hébergeur suisse, certifié écologique.
- **Hostinger** – hébergeur international low-cost.
- **AWS (Amazon Web Services)** – infrastructure cloud mondiale, très modulable.
- **Google Cloud Platform / Azure** – solutions professionnelles pour les grandes entreprises.
- **Vercel / Netlify** – hébergeurs modernes dédiés aux sites React, Next.js, SvelteKit ou Astro.

7. Hébergement, DNS et mise en ligne

7.4. Le rôle du DNS (Domain Name System)

- Le **DNS** est l'annuaire mondial du web.
  Il traduit un nom de domaine lisible par l'humain (www.monsite.fr) en **adresse IP** numérique (ex. 145.12.64.8) utilisée par les serveurs.

- Fonctionnement simplifié :
  1. L'utilisateur saisit monsite.fr.
  2. Le navigateur interroge le DNS.
  3. Le DNS renvoie l'adresse IP du serveur.
  4. Le navigateur établit la connexion HTTP ou HTTPS.

- Les DNS sont gérés par les **registrars** (Gandi, OVH, Google Domains, Infomaniak…) lors de la réservation du domaine.

7. Hébergement, DNS et mise en ligne

7.5. Les étapes de mise en ligne

1. **Choisir un hébergeur** et un plan adapté.
2. **Réserver un nom de domaine** (ex. : nomdusite.fr).
3. **Transférer les fichiers** du site via FTP ou interface web.
4. **Configurer la base de données** (souvent MySQL).
5. **Lier le domaine au serveur** via la zone DNS (enregistrements A, CNAME, MX).
6. **Tester le site en ligne** (liens, formulaires, compatibilité mobile).
7. **Installer un certificat SSL** (HTTPS) pour la sécurité.
8. **Mettre en place des sauvegardes et une stratégie de maintenance.**

7. Hébergement, DNS et mise en ligne

7.6. Maintenance et sécurité

- Une fois le site en ligne, il doit être **maintenu** et **surveillé** :
  - **Sauvegardes automatiques** (fichiers + base).
  - **Mises à jour régulières** du CMS, des thèmes et extensions.
  - **Surveillance de la performance** (Pingdom, UptimeRobot).
  - **Certificat SSL et HTTPS** permanents.
  - **Protection contre les attaques** (pare-feu, anti-DDoS, plugins de sécurité).
- Un site non maintenu devient rapidement vulnérable ou obsolète.

7. Hébergement, DNS et mise en ligne

7.7. L'hébergement écologique et responsable

- De plus en plus d'acteurs s'engagent dans le **Green Hosting** :
  hébergeurs utilisant des **énergies renouvelables**, des **data centers à refroidissement naturel**, et une **mesure de l'empreinte carbone** des sites.
- Exemples : **Infomaniak**, **Alwaysdata**, **PlanetHoster**, **Greengeeks**, **Scaleway**.
- L'impact environnemental du numérique est désormais un critère réel de choix technique.

> À retenir

- L'hébergement et la mise en ligne sont les **dernières étapes techniques** de tout projet web.
- Cela concrétise le travail du front-end, du back-end et du CMS, et garantit la disponibilité du site.
- Comprendre ces notions, c'est assurer la **pérennité, la sécurité et la visibilité** d'un projet.