Le Web

Le **Web** est un protocole d'échange de pages stockées sur des **serveurs web**, au format **HTML**.
Les **clients légers** (navigateurs web) se connectent aux serveurs en utilisant **HTTP**.

Les **liens hypertextes** permettent de naviguer de page en page. Le navigateur récupère les ressources et assure leur rendu graphique.

Serveur et Client Web

- **Serveur** : Exécute du code **PHP** ou **Servlet**, puis renvoie une page **HTML**.
- **Client** : Navigateur web qui affiche les pages reçues.

Processus d'échange

1. Le **client** demande la page au **serveur**
2. Le **serveur** génère ou forwarde la page demandée
3. Le **serveur** envoie la page HTML demandée au **client**
4. Le **navigateur** exécute un **code JavaScript** pour modifier la page HTML

Applications Web

Une **application web** est un **logiciel hébergé sur un serveur**, accessible via un réseau informatique par un navigateur.

Caractéristiques :

- Un seul développement pour tous les types d'appareils.
- Fonctionne sur tous les navigateurs.
- Accessible via les moteurs de recherche.

Types d'applications web

1. **Web statique** :
    - Affiche peu d'informations, généralement en **HTML et CSS**.
    - Peut contenir des **GIFs**.
    - Modification difficile : il faut éditer le code sur le serveur (**webmaster**).
2. **Web dynamique** :
    - Utilise une **base de données** pour charger les informations.
    - Contenu mis à jour dynamiquement à chaque accès.
    - Pas besoin d'intervenir sur le serveur pour mettre à jour les données.
3. **E-commerce** :
    - Gestion des **paiements électroniques** et des commandes.
    - Dispose d'un **panel de gestion** pour mettre à jour le contenu.
    - Peut être adapté en **application mobile** pour une meilleure expérience utilisateur.

4. **Web portail** :
    - Page d'accueil donnant accès à différentes sections (**forums, chat, moteur de recherche**…).
5. **Avec gestionnaire de contenu (CMS)** : (content management system)
   Permet de créer, gérer et modifier un site web sans avoir à coder.
    - Utilisé pour les sites nécessitant des mises à jour fréquentes.
    - Facile à gérer.
    - Exemples : **WordPress, Joomla, Drupal**.

Exemples d'applications web

- **Google Drive, Google Maps, Gmail, YouTube, Amazon**…
- Certaines applications web sont aussi accessibles via un mobile et un PC (ex : **WhatsApp Web, Telegram**).

Serveur et Client Web

- **Serveur Web**
    - Fournit le contenu demandé par le client.
    - Gère le traitement et la récupération des pages.
    - Stocke des données.
    - Prend en charge plusieurs protocoles : **HTTP, FTP, SMTP**.
    - Contient :
        - Un **SGBD** : Permet d'interroger et mettre à jour les données.
        - Un **LDAP** : Stocke les informations nécessaires à l'authentification.

- **Client Web**
    - Envoie des requêtes **HTTP** à un serveur.
    - Affiche les résultats reçus via un **navigateur web**.

  Navigateur Web
    - Utilise des protocoles comme **FTP, XML et HTML**.
    - HTTP possède plusieurs méthodes :
        - **GET** : Demande de contenu.
        - **HEAD** : Récupère uniquement l'en-tête de la réponse.
        - **POST** : Envoie des informations pour traitement.

Architecture des Applications Web

Architecture 4-Tier

1. **Présentation** : Clients légers (**ASP, JSP**) ou lourds (**Applet**).
2. **Applicative** : Règles métier, services et traitements.
3. **Objets métier** : Objets du domaine (ex : **Client, Facture**, etc.).

4. **Couche d’accès aux données** : Utilise un **SGBD-R**.

Bibliothèques et Frameworks

- **Bibliothèques** : Sous-programmes facilitant le développement.
- **Framework** : Ensemble de composants constituant une architecture logicielle.

Langages pour le Développement Web

- **ASP** : Intégré aux versions de Windows, permet le développement d’applications complexes.
- **PHP** : Concurrent principal d’ASP.
- **JSP** : Développé par **SUN**, permet d’insérer du code Java dans les pages HTML.

Frameworks Populaires

- **PHP** : Symfony, Laravel.
- **Python** : Django.
- **JavaScript** : Node.js, Angular.
- **Java** : Spring MVC, GWT, JSF.

Chapitre 2 : Architecture des applications web :

- **Frontend** : Interface graphique d’utilisateur
- **Backend** : Logique du métier
- **BD** : Contient les données utilisées, créées, supprimées ou modifiées par le backend, saisies par les utilisateurs à travers le frontend
- **Navigateur** : Permet aux utilisateurs d'interagir avec l'application, envoie et reçoit les requêtes HTTP

Caractéristiques d'une architecture :

- **Sécurité** : Disponibilité, intégrité, confidentialité, traçabilité, etc.
- **Élasticité** : Capacité à répondre au changement
    - Scalabilité verticale : Il s'agit d'augmenter la puissance d'un serveur unique en améliorant ses composants matériels.
    - Scalabilité horizontale : Elle consiste à ajouter plusieurs serveurs au système pour répartir la charge de travail.
- **Simplicité** : Code clair et organisé, réduire les dépendances pour un système léger

Types d'architectures :

1. Monolithique :

L'ensemble de l'application est développé et déployé en un seul bloc (centralisé).

- **Frontend :** (HTML, CSS, Javascript..)

- **Backend :** traitement et exécution des fonctionnalités
- **Base de données**

Cas d'utilisation :

- Petites applications
- Applications avec peu d'interactions entre différents modules.

Exemples :

WordPress, Boutique en ligne

Avantages :

- Simple à développer
- Performances optimisées (car peu de communications entre services)
- Déploiement unique

Inconvénients :

- Scalabilité difficile (car toute l'application doit être mise a jour, meme si ce n'est qu'une seule fonctionnalitée)
- Maintenance complexe

---

2. Architecture Microservice :

L'application est décomposée en plusieurs services indépendants. Les services communiquent entre eux via une API (REST, gRPC..)

Chaque service fonctionne indépendamment, mais il échangent les données entre eux.

Cas d'utilisation :

- Applications nécessitant une grande scalabilité
- Plateformes avec plusieurs fonctionnalités indépendantes

Exemples :

Netflix (diffusion, paiement, recommandation, etc.), Uber, Amazon, Spotify

Avantages :

- Scalabilité : chaque service peut être dimensionné indépendamment.
- Maintenance facile : un changement dans un microservice n'affecte pas le reste.
- Déploiement rapide
- Résilience : la panne d'un service n'affecte pas les autres.

Inconvénients :

- Complexité accrue (communication entre microservices)
- Consommation réseau
- Gestion des bases de données (chaque service a sa propre base de données)
- Cout supplémentaire

3. Architecture Serverless : **L'architecture Serverless, c'est quoi ?**

L’application est exécutée sans gestion directe des serveurs. Le cloud alloue dynamiquement les ressources en fonction de la demande. Les fonctions s’exécutent à la demande et s’arrêtent une fois terminées.

Cas d'utilisation :

- Applications avec une charge variable
- Exécution de tâches ponctuelles ou événementielles
- APIs sans infrastructure dédiée

Exemples : AWS Lambda, Google Cloud Functions, Azure Functions

Avantages :
✅ **Évolutivité automatique** : les ressources s’ajustent en fonction du trafic.
✅ **Coût optimisé** : facturation uniquement à l’usage.
✅ **Déploiement rapide** : pas de gestion d’infrastructure.
✅ **Maintenance réduite** : l’hébergeur gère les serveurs.

Inconvénients :
❌ **Latence à froid** : démarrage lent après une période d’inactivité.
❌ **Moins de contrôle** : dépendance aux fournisseurs cloud.
❌ **Limites de durée d’exécution** : chaque fonction a un temps maximal d’exécution.
❌ **Complexité accrue** : gestion des dépendances et de la persistance des données.

4. Architecture SOA :

Architecture où des services indépendants sont reliés par un ESB. Chaque service est conçu pour répondre à un besoin fonctionnel précis et peut être utilisé par plusieurs applications.

ESB : middleware qui facilite l'intégration et la communication entre services en assurant le routage

SOAP (Simple Object Access Protocol) : SOAP est un protocole de communication basé sur XML, utilisé pour échanger des messages entre services web.

Différence entre SOA et microservices :

Critère	SOA	Microservices
Granularité des services	Services plus grands, souvent plus complexes	Services plus petits et spécialisés
Communication	Via un **ESB**, souvent avec des protocoles lourds **SOAP**	Communication via des **API légères** (REST, gRPC).
Déploiement et scalabilité	Déploiement et scalabilité plus complexes	Déploiement indépendant, scalabilité horizontale
Autonomie des services	Services interdépendants	Services totalement autonomes
Gestion des données	Partage souvent une base de données commune	Chaque service peut avoir sa propre base de données

Cas d’utilisation :

Banques, Assurances, Systèmes gouvernementaux, télécoms…

Exemples : AirFrance, eBay..

Avantages :

- Réutilisation des services : Services partagés entre différentes applications
- Interopérabilité : Connecte facilement des systèmes hétérogènes grâce à des protocoles standards

Inconvénients :

- Complexité : ESB et communication via SOAP
- Latence : Retards dans les échanges

5. Architecture PWA :

Modèle hybride qui combine les applications web et mobiles natives.

- **Application Web** : Installé sur un serveur, et accessible via un navigateur, compatible avec tous les appareils, nécessite une connexion Internet, mise à jour automatique côté serveur.
- **Application Native** : Installée sur un appareil spécifique (Android, iOS, Windows), optimisée pour les performances, fonctionne hors ligne, nécessite des mises à jour manuelles via un store.

Cas d'utilisation : Réseaux sociaux, E-commerce, Actualité…

Exemples : Twitter Lite, AliExpress, Starbucks

Avantages :

- Accessible directement via navigateur
- Fonctionne hors ligne (stockage en cache)
- Pas besoin de mises à jour
- Performances élevées
- Multiplateformes

Inconvénients :

- Moins de visibilité sur les stores
- Dépendance au navigateur
- Compatibilité variable.

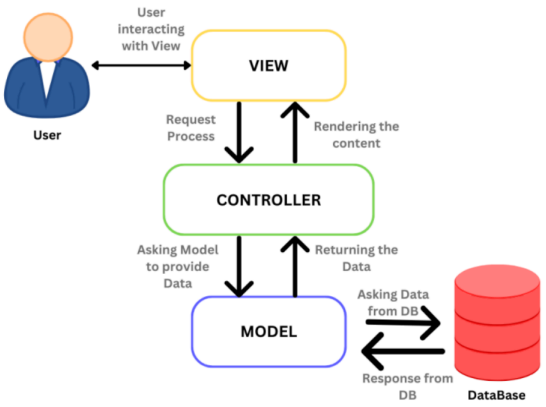
6. Architecture MVC :

Largement utilisée pour développer des applications web et desktop, elle divise l'application en 3 couches :

- **Modèle** : Gère les données et la logique métier
- **Contrôleur** : Gère les interactions entre vue et modèle

- **Vue** : Affiche les informations à l'utilisateur

Fonctionnement :



1. L'utilisateur interagit avec l'interface graphique (Vue).
2. La Vue envoie l'action au Contrôleur.
3. Le Contrôleur traite la demande et communique avec le Modèle.
4. Le Modèle récupère les données depuis la base et les envoie au Contrôleur.
5. Le Contrôleur met à jour la Vue avec les nouvelles données.

Utile pour les applications nécessitant une forte séparation entre interface et logique métier

Exemples :

Laravel, Spring MVC, Django

Avantages :

- Séparation des responsabilités
- Réutilisation du code
- Organisation claire

Inconvénients :

- Complexité accrue
- Temps d'apprentissage pour le développeur pour une bonne compréhension

---

7. Architecture client/serveur :

Modèle de communication ou un client envoie des requêtes à un serveur.

- **Client** : Demande un service
- **Serveur** : Répond aux requêtes
- **Réseau** : HTTP, TCP/IP, WebSocket

Types de tiers :

1-tiers : Tout est centralisé en un seul serveur (les bases de données, l'application et l'interface sont sur le même équipement)

2-tiers : Séparation entre la couche application et la base de données

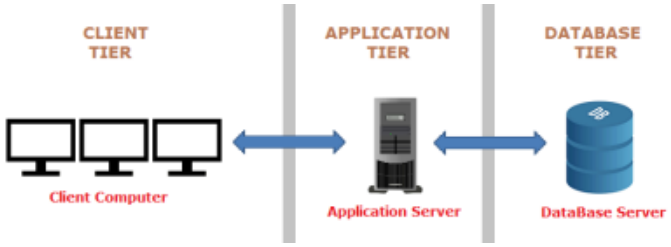
Avantages :

- plus sécurisé et plus performant que l'architecture 1-tier.

Inconvénients :

- Charge importante sur le serveur
- Difficulté à gérer un grand nombre d'utilisateurs

3-tiers : Les 3 couches (présentation, application et données) sont séparées en 3 tiers



Avantages :

- Séparation des responsabilités
- Meilleure scalabilité

Inconvénients :

- Complexe à mettre en oeuvre
- Nécessite plus de ressources matérielles

N-tiers :

Chaque couche a une responsabilité, permettant une meilleure scalabilité et maintenance des applications. Chaque couche communique avec les autres via API et protocoles réseaux.

Avantages :

- scalabilité
- sécurité

Inconvénients :

- Coût élevé
- Latence
- Complexité

Exemples N-tiers : Facebook, google maps, netflix..

Critère	Monolithique	Microservices	Serverless	SOA	PWA	MVC	Client-Serveur
Simplicité	✅	❌	✅	❌	✅	❌	✅
Scalabilité	❌	✅	✅	✅	✅	✅	✅
Maintenance	❌	✅	✅	✅	✅	✅	✅
coût	✅	❌	✅	❌	✅	✅	✅
Exemples	WordPress	Netflix	Uber	eBay	Twitter Lite	Django	Amazon

---

Chapitre 2 : Laravel

Laravel est un **framework PHP** gratuit et open-source qui suit l'architecture **MVC (Modèle-Vue-Contrôleur)**. Il est conçu pour faciliter le développement d'applications web en offrant une syntaxe élégante et des outils puissants. En 2024, il est l'un des frameworks **backend** les plus populaires.

1. Architecture de Laravel

Laravel suit une approche **MVC (Modèle-Vue-Contrôleur)** qui permet de séparer la logique métier, l'affichage et le contrôle des requêtes.

- **Modèle (Model)** : Représente la couche d'accès aux données. Il interagit avec la base de données pour récupérer, manipuler et enregistrer les informations.
- **Contrôleur (Controller)** : Gère la logique de l'application. Il reçoit les requêtes des utilisateurs, interagit avec le modèle pour récupérer les données nécessaires, puis les transmet à la vue.
- **Vue (View)** : Représente l'interface utilisateur. Elle reçoit les données du contrôleur et les affiche sous une forme lisible.

2. Schéma d'une requête client dans Laravel

1. L'utilisateur envoie une requête HTTP (`GET`, `POST`, `PUT`, `DELETE`).
2. Laravel fait le **routage de la requête** vers le **contrôleur** correspondant.
3. Le **contrôleur** analyse la requête et interagit avec le **modèle** si nécessaire.
4. Le **modèle** effectue des opérations sur la **base de données** et renvoie les données demandées.
5. Le **contrôleur** reçoit les données et les prépare pour l'affichage.
6. Les données sont envoyées à la **vue**, qui les affiche à l'utilisateur.

3. Outils et fonctionnalités clés de Laravel

3.1 Artisan

Interface en ligne de commande (CLI) de Laravel. Il permet d'automatiser des tâches comme la création de fichiers, la migration de base de données ou la génération de clés de chiffrement.

Exemples de commandes utiles :

- **Créer un modèle** :
  `php artisan make:model Post`
- **Créer un contrôleur** :
  `php artisan make:controller PostController`

3.2 Eloquent ORM

Eloquent est l'ORM (Object-Relational Mapping) intégré à Laravel, permettant d'interagir avec la base de données en utilisant des objets plutôt que d'écrire des requêtes SQL.

Exemple : Insérer un nouvel utilisateur

```
$user = new User;
$user->name = "Alice";
$user->email = "alice@example.com";
$user->save();  // Enregistre l'utilisateur dans la base de données
```

Grâce à Eloquent, les opérations **CRUD (Create, Read, Update, Delete)** sont simplifiées.

3.3 Blade

Blade est le moteur de template de Laravel, permettant d'intégrer du code PHP directement dans les fichiers de vue.

Exemple : Affichage d'un message personnalisé

```
<h1>Bonjour, {{$name}}</h1>
```

Blade facilite la gestion des conditions (`@if`), des boucles (`@foreach`) et l'inclusion de fichiers (`@include`).

3.4 Composer

Composer est un gestionnaire de dépendances PHP utilisé par Laravel pour installer et gérer les bibliothèques et extensions.

Exemple : Installer Laravel via Composer

```
composer create-project --prefer-dist laravel/laravel nom_du_projet
```

PHP

- **HTML s'occupe du formatage, car PHP ne le fait pas**
- **Si un fichier HTML contient du PHP, il doit être renommé en .php ou .phtml**
- **Le code PHP se place dans le body du fichier HTML**

Affichage en PHP

- **`echo` : afficher un message**
- **`<br>` : retour à la ligne**

Exemple

```php
<?php
echo "I like pizza <br>";
echo "It's really good";
// Commentaire sur une ligne
/* Commentaire
   sur plusieurs
   lignes */
?>
```

PHP peut contenir du HTML

```php
<!DOCTYPE html>
<html lang="en">
<head>
    ...
</head>
<body>
    <?php
    echo "I like pizza";
    ?>
    <br>
    <button>Order pizza</button>
</body>
</html>
```

Variables en PHP

PHP gère plusieurs types de variables : String, Integer, Float, Boolean

Exemple

```php
<?php
```

```php
$nom = "Ranim Gouider";
echo $nom;
?>
```

Concaténation

```php
<?php
$nom = "Ranim Gouider";
$age = 23;

echo "Hello {$nom} <br>";
echo "Hello " . $nom . "!";
echo $nom . " a " . $age . " ans";

$prix = 50;
echo "The item is worth \${prix}";
?>
```

Opérations mathématiques

```php
<?php
$x = 5;
$y = 10;

$somme = $x + $y;
$sub = $x - $y;
$multip = $x * $y;
$div = $x / $y;

echo $somme . "<br>";
echo $sub . "<br>";
?>
```

Autres opérateurs

- `++` : ajoute 1
- `--` : soustrait 1
- `**` : puissance
- `%` : reste de division
- `+=, -=, *=, /=`

Priorité des opérateurs

1. `()`
2. `**`
3. `* / %`
4. `+ -`

Méthodes PHP : $_GET et $_POST

`$_GET` et `$_POST` permettent de récupérer les données envoyées via un formulaire HTML.

Méthode GET (visible dans l'URL, peu sécurisée)

```php
<form action="index.php" method="get">
    <label>Username:</label>
    <input type="text" name="username">
    <input type="submit" value="Log in">
</form>

<?php
echo $_GET["username"];
?>
```

Méthode POST (sécurisée, données non visibles)

```php
<form action="index.php" method="post">
    <label>Username:</label>
    <input type="text" name="username">
    <input type="submit" value="Log in">
</form>

<?php
echo $_POST["username"];
?>
```

GET pour récupérer des données.
POST pour envoyer des données sensibles ou volumineuses.

Conditions en PHP

if - else

```php
<?php
$age = 15;

if ($age >= 18) {
```

```
    echo "You may enter the site";
} else {
    echo "Go away!";
}
?>
```

if - elseif - else

```
<?php
$age = 15;

if ($age >= 18) {
    echo "You may enter the site";
} elseif ($age <= 0) {
    echo "You don't exist";
} elseif ($age >= 100) {
    echo "Too old";
} else {
    echo "Go away!";
}
?>
```

Opérateurs logiques

- `&&` : ET
- `||` : OU
- `!` : NON

Exemple

```
<?php
$temp = 25;

if ($temp >= 15 && $temp <= 30) {
    echo "Weather is good :)";
} else {
    echo "Weather is bad!";
}
?>
```

Switch Case

```
<?php
```

```php
$grade = "A";

switch($grade) {
    case "A":
        echo "Good job";
        break;
    case "B":
        echo "Nice";
        break;
    case "C":
        echo "Average";
        break;
    case "D":
        echo "Bad!";
        break;
    case "F":
        echo "Fail";
        break;
    default:
        echo "Not valid";
        break;
}
?>
```

Boucles en PHP

For Loop

```php
<?php
for ($i = 0; $i < 5; $i++) {
    echo $i . "<br>";
}
?>
```

While Loop

```php
<?php
$counter = 0;
while ($counter <= 10) {
    echo $counter . "<br>";
    $counter++;
} ?>
```

Les tableaux (arrays)

```php
<?php
$foods = array("pizza", "burger", "chips", "brika");

echo $foods[0] . "<br>";

foreach ($foods as $food) {
    echo $food . "<br>";
}

// Ajouter un élément
array_push($foods, "ananas");

// Supprimer le dernier élément
array_pop($foods);

// Supprimer le premier élément
array_shift($foods);

// Inverser un tableau
$reverseFoods = array_reverse($foods);
?>
```

Tableau associatif

```php
<?php
$tab = array("USA" => "Washington DC", "JP" => "Kyoto", "SK" =>
"Seoul", "TN" => "Tunis");

foreach ($tab as $key => $value) {
    echo "{$key} = {$value} <br>";
}

// Modifier une valeur
$tab["USA"] = "Las Vegas";

// Ajouter un élément
$tab["FR"] = "Paris";
?>
```

Formulaires en PHP

Radio Button

```
<form action="file.php" method="post">
    <input type="radio" name="cc" value="Visa"> Visa <br>
    <input type="radio" name="cc" value="Mastercard"> Mastercard <br>
    <input type="radio" name="cc" value="American Express"> American
Express <br>
    <input type="submit" name="OK" value="OK">
</form>
```

Checkbox

```
<form action="file.php" method="post">
    <input type="checkbox" name="Pizza" value="pizza"> Pizza <br>
    <input type="checkbox" name="Borgir" value="Borgir"> Burger <br>
    <input type="checkbox" name="Tacoth" value="Tacoth"> Tacos <br>
    <input type="submit" name="OK" value="OK">
</form>
```

Fonctions en PHP

```
<?php
function happy_birthday($name, $age) {
    echo "Happy Birthday {$name}! <br>";
    echo "You are {$age} years old!";
}

happy_birthday("Ranim", 23);
happy_birthday("Rihem", 24);
?>
```

Liste déroulante

```
<form name="Saisie" action="file.php" method="post">
    <select name="choix">
        <option value="choix 1">Choix 1</option>
        <option value="choix 2">Choix 2</option>
        <option value="choix 3">Choix 3</option>
    </select>
    <input type="submit" name="OK" value="OK">
</form>
```