Table of Contents

Introduction

A l'occasion de la deuxième édition du Forum du Developement et Réseau organisé par le Club des Jeunes Programmeurs les **17 et 18 Mai 2024**, nous avons préparé ce cours qui introduit les notions de bases et des outils constituant le b.a-ba du développeur Web, d'où l'intitulé **Web Roadmap**.

Ce cours loin d'être une exhaustivité présente toutes fois des contenus assez riches qui peuvent être très utiles pour le débutant afin de lui servir de guide dans le vaste domaine qu'est le développement web.

Pour terminer, nous espérons que ce cours servira le plus grand nombre de personnes et que vous profiterez de chaque miette de son contenu.

Happy Coding ❤️‍🩹

NET ESSENTIALS

Savez-vous en réalité ce qui se passe lorsque vous tapez sur la barre de recherche de votre navigateur l'URL du site du Club des Jeunes Programmeurs https://club-jp.com ?

Votre appareil est en train en coulisse de demander d'afficher le contenu d'une page web qui se trouve sur une machine distante.

Tout d'abord, cette phrase soulève des questionnements:

- Mon appareil demande d'afficher ?
- Une page web ? Mais une page web c'est quoi ?
- Une machine ? Elle est à qui cette machine ?

Nous allons à travers les lignes qui suivent, expliquer dans les grandes lignes les principes de cette communication qui est au coeur même du web, et en général d'Internet.

Quoi, Web et Internet c'est différent ?

Oui !

Le Web est un des nombreux services offerts par Internet. les autres services sont par exemple la messagerie email, le transfert de fichiers, etc.

Internet par contre c'est le réseau des réseaux. Littéralement cela signifie **Interconnected Networks**, sachant qu'un **réseau** est une communication entretenue entre deux et plusieurs machines.

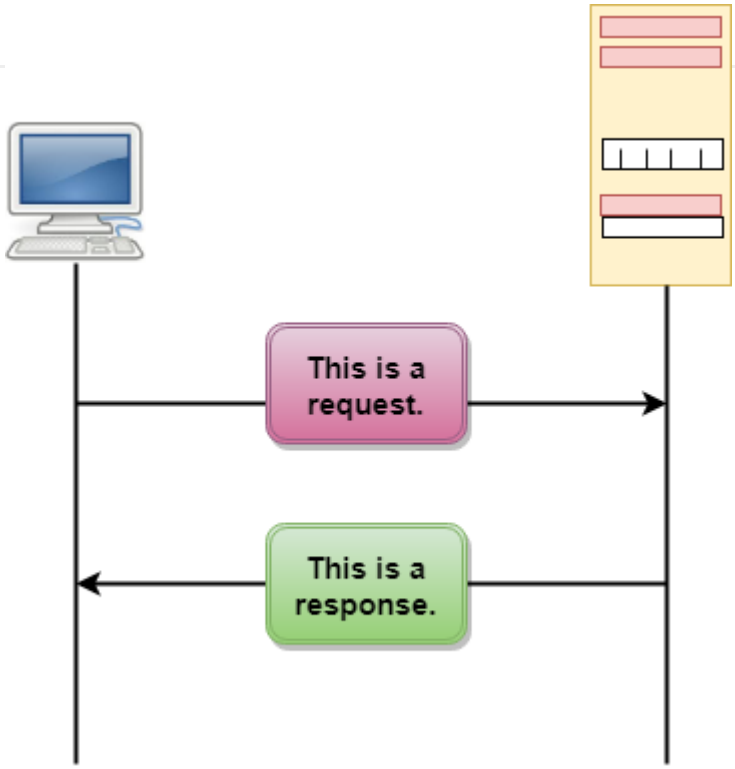
Au coeur du Web, nous avons le *protocole* HTTP.

Un **protocole** c'est un ensemble de normes et de procédés qui permettent de réguler une communication.

Que dit ce protocole ?

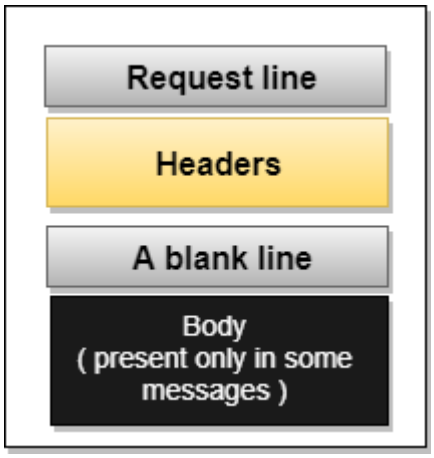
Le protocole HTTP est celui qui régit les communication sur le Web, à savoir, il s'agit de l'architecture Client / Serveur.

Le **client** est le demandeur tandis que le **serveur** est la machine qui est interrogée.

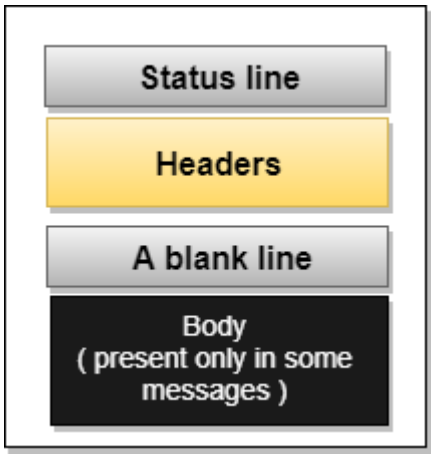


Les machines qui communiquent sur le web échangent des messages, qui peuvent être de deux nature: **request** ou bien **response**.

Le client envoie une demande (request)



Tandis que le serveur réponds (response)



L'URL

Ce qui a rendu le Web intéressant et de facto l'Internet n'est rien d'autre que les hyperliens que l'on construit avec les URL.

Les **liens** permettent de relier une page Web à une autre.

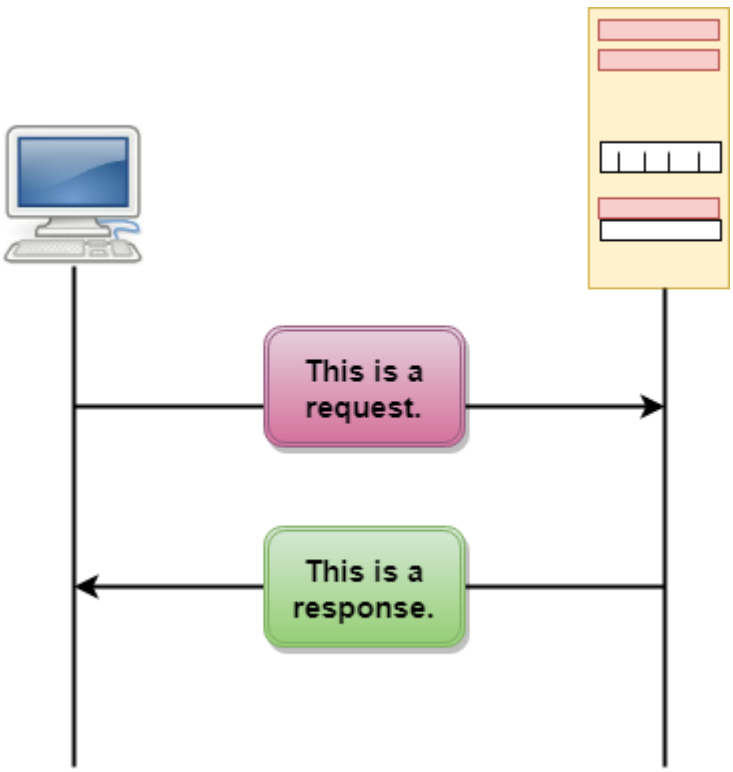
Une **URL** (Uniform Resource Locator) par contre permet de localiser une resource ou bien du contenu (fichier) sur Internet.

Voici quelques exemples:

- https://club-jp.com/
- https://univ-labe.edu.gn/
- https://univ-labe.edu.gn/wp-content/uploads/2024/03/125A4402-2048x1365.jpg

Client / Serveur

Maintenant que nous savons cela, reprenons cette image au début:



Vous devez retenir ceci:

Lorsque vous décidez de lancer le site du Club des Jeunes Programmeurs, indirectement, votre appareil demande à une certaine machine sur laquelle le site est stoqué de charger cette page.

La machine va donc répondre en envoyant le contenu de la page et puis votre appareil (ou particulièrement votre navigateur) va se charger de l'interpréter et afficher le résultat.

Voici donc décrit comment fonctionne le Web.

Mais cependant une question réside, comment on met cela en place ?

Eh bien, à travers cette série de formation, nous allons apprendre les technologies qui vont vous permettre d'arriver à cet objectif.

Plus spécifiquement, nous allons partager ensemble des techniques et technologies utilisées pour faire du développement web.

Alors vous êtes chaud ?

Rendez-vous dans le chapitre suivant où nous allons parler de la programmation web, et particulièrement du HTML et du CSS.

HTML

Le HTML est un langage de *balisage* qui permet de définir la structure d'un contenu.

Un document HTML est ainsi une suite d'**éléments** utilisé pour encadrer le contenu afin de les faire apparaître ou les faire se comporter d'une certaine façon.

Exemple Si nous prenons le contenu:

```
Quelle belle mélodie !
```

Si on veut que la ligne ait son propre paragraphe, on peut utiliser les **balises** qui correspondent: `<p>` pour encadrer le contenu:

```
<p>Quelle belle mélodie !</p>
```

Une balise ? Un élément ?

Oui, les balises sont utilisées pour donner de la **valeur** à un contenu, un **sens** en quelque sorte.

Structure d'un élément HTML

Reprenons le snippet de la section précédente.

```
<p>Quelle belle mélodie !</p>
```

On peut identitfier trois parties dans ce bout de code.

```
flowchart
    subgraph Element HTML
        direction TB
        A[&lt;p&gt;]
        B[Quelle belle mélodie]
        C[&lt;/p&gt;]

        A --- 1[Balise ouvrante]
        B --- 2[Contenu]
        C --- 3[Balise ouvrante]
        end
        direction RL
```

Les éléments peuvent aussi parfois avoir des attributs comme ceci:

```
<p class="note" id="com12">Quelle belle mélodie !</p>
```

Un attribut contient des informations supplémentaires à propos de l'élement mais qui ne sont pas affichées. **class** c'est le nom de l'attribut et **note** correspond à sa valeur.

Certains attributs n'ont pas de valeur, comme **required** (Formulaires).

Suivez ces règles:

1. Les attributs sont séparés par des espaces.
2. La valeur est entre les guillemets doubles (`"`) ou simples (`'`)

Imbrication des éléments

Il s'agit du fait de placer un élément à l'intérieur d'un autre.

Par exemble pour accentuer sur le mot **belle** du paragraphe précédent, on peut faire comme suit:

```
<p>Quelle <strong>belle</strong> mélodie !</p>
```

Ce pendant, il faut observer la règle que le premier élément ouvert sera le dernier à être fermé. (LIFO)

Utilisez ce diagramme pour comprendre:

```
flowchart
    p --- 1[Quelle]
    p --- 2[strong]
    2 --- 3[belle]
    p --- 4[mélodie !]
```

Eléments vides

Il existe aussi des éléments vides. Par exemple pour insérer une image, la balise c'est `img`

```
<img src="images/firefox-icon.png" alt="Mon image de test"/>
```

Anatomie d'un document HTML

Cette théorie qu'on vient de voir, s'appliquera à l'ensemble des éléments HTML. Cependant, sur page Web, on n'aura pas qu'un seul élément HTML. C'est pourquoi, il est important d'voir à quoi ressemble la structure d'une page Web.

Analysons le code suivant:

```
<!doctype html>
<html>
    <head>
        <title>Une page Web</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>

    <body>
        <img src="images/renard.jpg" alt="L'image d'un renard">
    </body>
</html>
```

On y trouve ce qui suit:

```
flowchart
    subgraph 1
        A[doctype]
        end
    subgraph 2
        direction TB
        B[html]
        B --> C[head]
        B --> D[body]
        C --> E[title]
        C --> F["meta (charset)"]
        C --> G["meta (viewport)"]
        D --> H[q]
        end
```

La suite ?

Maintenant qu'on a la base, approfondissons la notion de balises.

Avec le langage HTML, on peut donner différentes valeurs à du texte. Commençons par le début, les **titres**.

- Les éléments **titre** Permettent d'indiquer qui sont les titres ou les sous-titres au sein de la page. Pour ce faire, on utilise les éléments `<h1>` - `<h6>` .

  h1 a plus de valeur que *h6*.

  Si on prend exemple sur ce chapitre du cours jusqu'à ce niveau, on aura:

  ```
  <h1>Apprendre les Fondamentaux du Web avec un projet guidé de A à Z</h1>
  <h2>Notions de base en HTML</h2>
  <h3>Une balise ? Un élément ?</h3>
  <h4>Structure d'un élément HTML</h4>
  ```

  > N'utilisez pas les niveaux de titres pour la mise en forme. Vraiment !

- Les paragraphes Le texte dans un document HTML doit être wrappé dans un paragraphe. Il sert à baliser du texte qui se trouve sur la page Web.

  ```
  <p>Voici un paragraphe simple</p>
  ```

- Les listes On peut rencontrer dans le contenu d'une page Web des liste. Le langage HTML a des éléments dédiés pour décrire les listes, ordonéées ou non.

  - Listes ordonnées (ordered list: `ol`)
  - Listes non ordonnées (unordered list: `ul`)
  - Chaque élément de liste est un *list item (`li`)*

    Exemple:

```
<p>En HTML, il existe deux sortes de listes:
<ul>
    <li>Listes ordonnées</li>
    <li>Listes non ordonnées</li>
</ul>
```

- Liens Les liens sont une partie essentielle du Web. Pour ajouter un lien en HTML on utilise l'élément `<a>`.

```
<p>Visitez mon <a href="https://facebook.com/moustaphaotf">profile Facebook</a>.</p>
```

CSS

Alors, c'est bien curieux ! Mais voilà, ils ont créé le langage **CSS**. Qui permet de créer des feuilles de styles qu'on peut appliquer à notre squelette HTML afin de lui donner vie, de la forme, et des couleurs.

Il s'agit du **World Wide Web Consortium** (W3C) qui a la responsabilité de tout ce qui est spécification Web.

Le langage CSS nous permet donc de donner des styles à notre contenu HTML.

Nous allons y aller petit à petit pour découvrir les propriétés de styles fondamentales. Mais avant, il est important de pouvoir:

- Lier deux fichiers HTML et CSS.
- Comprendre les sélecteurs.

Liaison des fichiers

Dans notre HTML, on peut indiquer quels fichier de style s'appliquent à notre page.

Pour cela, on utilise la balise `link` placée dans l'élément `head` de la page.

```
    <!doctype>
    <html>
        <head>
            <title>Le langage CSS</title>
            <meta charset="utf-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
+           <link rel="stylesheet" href="styles.css" >
        </head>
        <body>
            <h1>Apprenons les styles en CSS</h1>
        </body>
    </html>
```

Enregistrons le contenu et puis finalement, dans le même dossier que notre fichier HTML, on crée un fichier `styles.css` .

Notions de sélecteurs

Maintenant que nous pensons que les fichiers HTML et CSS sont reliés, vérifions en appliquant notre premier style.

Dans le fichier CSS, on ajoute ce bout de code:

```
body {
    background-color: green;
}
```

Enregistrons le fichier CSS et puis on va ouvrir le fichier HTML dans le navigateur.

Si tout est bien fait, on devrait avoir une nouvelle couleur à l'arrière plan: le **vert**.

Voilà, on vient comme ça d'écrire notre premier code CSS.

> **Note :** Il est inutile d'ouvrir le fichier CSS dans le navigateur. Nous avons créé le fichier HTML et nous avons ajouté une ligne pour le relier au code CSS. Du coup, il va importer les styles qui sont dans le fichier CSS pour se les appliquer.

Expliquons brièvement le code qu'on a écrit tout de suite.

En CSS, on a les sélecteurs, les propriétés et les valeurs; c'est tout !

```
flowchart
A[body] --- B[Sélecteur]
C[background-color] --- D[Propriété]
E[green] --- F[Valeur]
```

On va s'intéresser aux sélecteurs, il en existe plusieurs, mais on verra quelques uns seulement.

- Sélecteurs d'éléments comme `body` , `p` , `div` , `ul` , etc. On prend juste le nom de l'élement que l'on cible. Les propriétés seront appliqués à tous les éléments de la page qui sont de même type. Par exemple

  ```
  p{
      color: red;
  }
  ```

  va colorier tous les paragraphes du document en rouge.

- Les sélecteurs de classe.

  En HTML, on peut créer des éléments en leur ajoutant un attribut `class` .

  ```
      <body>
          <h1>Les sélécteurs</h1>
          <p class="highlight">Lorem ipsum</p>
          <p>Dolor sit</p>
          <p class="highlight">Amet consectetur</p>
      </body>
  ```

  Grâce à cet attribut, on pourra les cibler en CSS, comme suit:

  ```
  .highlight {
      color: red;
      font-weight: 700;
  }
  ```

  Voilà !

  On peut même :

  ```
  p.highlight {
      color: red;
      font-weight: 700;
  }
  ```

  pour dire qu'on ne s'intéresse qu'aux paragraphes qui ont la classe `highlight`

> **Note:** Remarquez que la classe est collée au nom de l'élément.

Si on veut styliser le paragraphe qui n'a pas la classe `highlight`, cibler juste avec `p` ne sera pas suffisant.

Pour ce faire, on peut:

```
p:not(.highlight) {
    color: red;
    font-weight: 700;
}
```

- Les sélecteurs d'identifiant !
  Si on ajoute un attribut `id` à notre élément, alors on peut le cibler en CSS avec `#<nom-identifiant>`.

  Par exemple:

```
<body>
    <h1>Les sélécteurs</h1>
    <p>Dolor sit</p>
    <p id="second">Amet consectetur</p>
</body>
```

```
#second {
    color: red;
    font-weight: 700;
}
```

> **Note :** Un identifiant est unique par page HTML.

Et ainsi de suite, il y en a plein d'autres sélecteurs comme :

- Sélecteur universel :

```
/* Pour cibler tous les éléments de la page, quels qu'ils soient */
* {
    margin: 0;
}
```

- Sélecteur de descendants:

```
/* Pour cibler toutes les cellules d'un tableau */
table td {
    border: 1px solid black;
}
```

- Sélecteurs d'enfants directs:

```
/* Pour cibler tous les paragraphes qui sont des enfants directs d'une div */
div > p {
    text-align: center;
}
```

- Sélecteurs de voisins

```
/* Pour cibler tous les paragraphes qui sont positionnés après des h2 */
h2 + p {
    margin-top: 50px;
    border-top: 2px dashed gray;
}
```

> **Note :** On peut grouper les sélecteurs en les séparant par une virgule.

Valeurs spéciales

En CSS, il serait intéressant qu'on révise un peu quelques valeurs intéressantes:

- Les mesures
- Les couleurs

Les mesures

Pour le cas des **mesures**, nous avons plusieurs unités telles que:

1. Les unités statiques
   - **pixel** (`px`)
   - **centimètre** (`cm`)
   - **inch** (`in`)
2. Les unités relatives
   - **pourcentage** (`%`)
   - **em** (relatif à la taille de police de l'élement parent.)
   - **rem** (relatif à la taille de police de l'élement `:root` ou `body`)
   - **vh** (relatif à hauteur du viewport)
   - **vw** (relatif à la largeur du viewport)

Cependant pour le cadre de ce cours, on se contentera de trois unités: `px` , `rem` et `em` .

Les couleurs

Pour ce qui est des couleurs, il existe plusieurs façon de les exprimer également.

> Basiquement, on peut obtenir n'importe quelle couleur en additionnant des couleurs fondamentales : `rouge` , `bleu` et `vert` .

On va retenir trois façons pour donner une couleur (mais il en existe d'autres.)

- A partir de son nom Oui, des nom comme `blue` , `black` , `white` , `red` , `yellow` , `green` , `teal` , `purple` , `gray` , `turquoise` , `pink` , `violet` , etc. Une liste non exhaustive se trouve sur cette page.
- A partir de son **code rgb** Pour le code **RGB**, il suffit de connaitre les variation de `rouge` , `bleu` et `vert` de la couleur pour obtenir la couleur que l'on veut. La quantité minimale pour chaque variation est **0** tandis que la quantité maximale est **255**.

On va prendre quelques exemples

|Couleur|Code RGB| |-|-| |Noir| `rgb(0, 0, 0)` | |Blanc| `rgb(255, 255, 255)` | |Bleu| `rgb(0, 0, 255)` | |Rouge| `rgb(255, 0, 0)` | |Vert| `rgb(0, 255, 0)` | |Jaune| `rgb(255, 255, 0)` |

- A partir de son **code hexadécimal** Finalement, il suffit de prendre la correspondance hexadécimale de chaque quantité du code **RGB** pour former une couleur en mode hexa. On va reprendre le tableau précédent.

  |Couleur|Code Hexa| |-|-| |Noir| `#000000` | |Blanc| `#FFFFFF` | |Bleu| `#0000FF` | |Rouge| `#FF0000` | |Vert| `#00FF00` | |Jaune| `#FFFF00` |

```
> Note: Pour les valeurs **hexa**, naturellement, la quantité minimale est **00** tandis que la quantité maximale est **FF
```

Texte mielleux

Après cet apparté sur les sélecteurs, les valeurs particulières, nous allons nous intéresser à comment on va enrichir notre texte. Pour cela, nous aborderons nombre de propriétés ayant trait au texte.

Propriété	Description
`font-family`	Pour changer la police d'écriture
`font-weight`	Pour mettre le poids de la police (gras)
`font-size`	Modifier la taille de police
`text-align`	Changer l'allignement du texte
`text-decoration`	Pour la décoration du texte (soulignement)
`text-transform`	Changer la casse (majuscule, minuscule)
`color`	Pourla couleur de texte

Système de boites

En HTML, tout est une boite. Mais on classe les éléments en deux: **inline** et **block**.

> **Inline:** Ceux qui s'insèrent au flux et ne cause pas de décalage en occupant toute la largeur de l'écran.
>
> **Block:** Ceux qui s'insèrent au flux et causent un décalage en occupant toute la largeur qu'on leur attribue.

Et y a certaines propriétés assez intéressantes pour les boîtes:

- Les bordures (`border`)
- Les marges internes (`padding`)
- Les marges externes (`margin`)

JS

Le JavaScript est un langage de programmation qui permet de rendre des pages Web dynamiques. Ce dynamisme se manifeste très souvent par des comportements programmables qui permettent à la page Web de se comporter différemment lorsque telle ou telle autre chose survient.

L'exemple le plus simple que l'on va faire sera de créer une page dont l'arrière plan change à chaque clic.

Premièrement examinons les différents briques de bases pour pouvoir construire un tel programme.

Tout d'abord, les outils nécessaires:

- Un **éditeur** pour écrire notre code source
- Un **navigateur** pour interpréter (afficher)

Les bases du JS

Les variables

Une variable sert à stocker une valeur utile pour le programme lors de son fonctionnement.

Pour créer une variable, nous utilisons le mot clé `let` .

Il existe plusieurs type de données:

- Numérique
- Booléen
- String
- Object
- Array

Pour résumer, retenons ceux-là ! Ils feront le taff presque 90% du temps.

```
let sentence = "Bienvenue au FODR"
console.log(sentence)

let name = "Moustapha"
let age = 10
console.log(name, "a", age, "ans")

let values = [12, 45, 89]
console.log(values[0])

let person = {
    name: 'Moustapha',
    city: 'Mamou',
    dob: '1999-05-17',
}

console.log(person.age)
console.log(person['city'])
console.log(person.dob)
```

Alors, la console est certes un endroit où l'on peut expérimenter, mais ce n'est pas adapté pour écrire des script.

Alors, nous allons résoudre notre premier projet, changer dynamiquement l'arrière plan.

Source Code: Random Color Generator

Conditions & Boucles

Lorsque vous programmez en JS, il arrive très souvent de vouloir automatiser des tâches, répéter et controller le flux d'exécution.

Les **conditions** permettent de controller le flow de notre programme, alors il existe principalement trois façons:

- `if else`
- `switch`
- `ternary`

Les opérateurs de comparation et aussi logiques permettent d'exprimer les conditions.

Les **boucles** permettent quant à elles de repéter des actions, ou alors un bloc d'instructions.

Exemple

```
// prenons une liste de personnes:
const persons = [
    {
        nom: 'Moustapha',
        dob: '12-12-1999',
    },
    {
        nom: 'Oury',
        dob: '12-05-2000',
    },
    {
        nom: 'Aïssatou',
        dob: '05-20-2001',
    },
    {
        nom: 'Moustapha',
        dob: '17-07-2000',
    },
]

// Liste de tous le monde
for(let i = 0; i < persons.length; i++)
{
    console.log(`${i + 1}: ${persons[i].nom} (${persons[i].dob})`)
}

// Liste des personnes nées à partir de 2000
for(let i = 0; i < persons.length; i++)
{
    if(persons[i].dob >= '01/01/2000')
    {
        console.log(`${i + 1}: ${persons[i].nom} (${persons[i].dob})`)
    }
}
```

Maintenant essayons de réaliser un petit programme qui va simuler une roulette.

Code Source: TP Roulette

Browser APIs

Les navigateurs de nos jours embarquent des pour la plupart des API qui permettent d'effectuer des opérations beaucoup plus complexes qu'il aurait été impossible ou bien très difficile les années précédentes.

- DOM
- Canva & WebGL
- Storage
- Géolocalisatin
- Fetch
- etc.

Git & GitHub

Git

Git est un système de contrôle de version distribué qui permet de suivre les modifications apportées aux fichiers et de collaborer avec d'autres développeurs.

Installation : Git peut être installé via le site officiel git-scm.com.

Configuration initiale :

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

Commandes de base :

git init : Initialise un nouveau dépôt Git. **git clone :** Clone un dépôt distant. git status : Vérifie l'état des fichiers dans le répertoire de travail. **git add :** Ajoute des fichiers à l'index (staging area). **git commit -m "Message" :** Enregistre les modifications dans l'historique du projet. **git push :** Envoie les commits vers un dépôt distant. **git pull :** Récupère et fusionne les modifications depuis un dépôt distant. **git branch :** Liste, crée ou supprime des branches. **git checkout :** Change de branche ou restaure des fichiers. **git merge :** Fusionne une branche dans la branche courante. **git log :** Affiche l'historique des commits.

GitHub

GitHub est une plateforme de gestion de projets et de partage de code qui utilise Git pour le contrôle de version.

Création de compte : Inscrivez-vous sur GitHub et créez un dépôt.

Interaction avec GitHub :

- Créer un dépôt :

Via l'interface web ou avec git init suivi de git remote add origin .

- Collaborer :

Fork : Copiez le dépôt d'un autre utilisateur sur votre compte. Pull Request (PR) : Proposez des modifications sur un dépôt. Issues : Suivre les bogues et les fonctionnalités. Actions : Automatiser les workflows avec CI/CD.

- Cloner un dépôt :

```
git clone https://github.com/nom-utilisateur/nom-depot.git
```

- Synchronisation :

git remote add origin : Ajoute un dépôt distant. **git push -u origin master :** Pousse les commits sur la branche master. **git fetch :** Récupère les changements sans fusionner. **git pull :** Récupère et fusionne les changements.

Concepts avancés

Branches : Utilisées pour travailler sur des fonctionnalités indépendantes sans affecter la branche principale.

```
git branch nouvelle-branche
git checkout nouvelle-branche
```

Merge : Combinaison des changements de différentes branches.

```
git checkout master
git merge nouvelle-branche
```

Rebase : Applique une suite de commits sur une autre base.

```
git rebase master
```

Résolution de conflits : Parfois nécessaire lors des fusions ou des rebases.

Stash : Sauvegarde temporaire des modifications non commises.

```
git stash
git stash apply
```

En maîtrisant Git et GitHub, vous pouvez gérer efficacement les versions de votre code et collaborer avec d'autres développeurs de manière fluide.

Conclusion

En conclusion, Git et GitHub sont des outils essentiels pour la gestion de versions et la collaboration en développement logiciel. Git permet de suivre l'historique des modifications et de travailler en parallèle sur différentes fonctionnalités grâce à son système de branches. GitHub enrichit cette expérience en offrant une plateforme collaborative où les développeurs peuvent partager leur code, gérer des projets, et automatiser des workflows. En combinant Git et GitHub, vous pouvez non seulement améliorer votre productivité, mais aussi garantir une meilleure qualité de code grâce à une gestion structurée et collaborative.

Hosting

Le **hosting** autrement dit déploiement est une étape dans le processus de développement d'application web.

Il s'agit de mettre notre application en production afin que les internautes puissent y accéder librement, à un click près.

Pour pouvoir héberger votre site internet, la première des choses à faire donc c'est d'abord vous assurer qu'il fonctionne très bien en local sur votre ordinateur.

Puis aller **à la recherche des hébergeurs**.

L'hébergement est un service offert par des entreprises pour vous louer un ordinateur sur lequel faire tourner votre code source **24/7**, moyennant un paiement.

Il existe de nombreuses entreprises qui offrent ce service parmi lesquelles on peut citer:

- Hostinger
- AWS
- Vercel
- Netlify
- LWS
- Github Pages
- etc.

Vous pouvez toutes fois obtenir une offre gratuite chez ces hébergeurs afin de tester vos applications et vos idées de projets simplement.

D'ailleurs si vous créez des sites statiques, vous pouvez les héberger facilement sur Github, grâce à leur service Github Pages.

Introduction à React

React est une bibliothèque JavaScript développée par Facebook pour construire des interfaces utilisateur (UI). Elle permet de créer des applications web interactives et dynamiques.

Concepts de Base

Composants :

Les composants sont les blocs de construction de base d'une application React. Ils peuvent être des classes ou des fonctions.

Chaque composant peut avoir son propre état et ses propres propriétés (props).

JSX (JavaScript XML) :

Une syntaxe qui permet de décrire la structure de l'UI de manière déclarative.

Il ressemble à HTML, mais permet d'intégrer des expressions JavaScript.

Props :

Abréviation de "properties", les props sont des arguments passés aux composants pour les configurer.

Elles sont immuables, c'est-à-dire qu'elles ne peuvent pas être modifiées par le composant qui les reçoit.

State :

Le state est un objet qui contient des données spécifiques à un composant et qui peuvent changer au fil du temps.

Le state est géré au sein du composant et peut être modifié en utilisant setState (dans les composants de classe) ou la fonction useState (dans les composants fonctionnels).

Création d'un Composant de Base

```
import React from 'react';


function MyComponent(props) {
  return <h1>Hello, {props.name}!</h1>;
}
export default MyComponent;
```

Composants Fonctionnels et Hooks

Les composants fonctionnels sont des fonctions JavaScript simples qui retournent des éléments React.

Les hooks permettent d'ajouter des fonctionnalités d'état et de cycle de vie aux composants fonctionnels.

- Exemple d'utilisation du hook useState :

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
  export default Counter;
}
```

- Exemple d'utilisation de useEffect :

```
import React, { useEffect, useState } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(seconds => seconds + 1);
    }, 1000);

    return () => clearInterval(interval); // Cleanup
  }, []);

  return <p>Seconds: {seconds}</p>;
}
export default Timer;
```

Gestion des Evénements

Les événements en React sont gérés de manière similaire aux événements DOM, mais avec une syntaxe légèrement différente.

```
function MyButton() {
  function handleClick() {
    alert('Button clicked!');
  }

  return (
    <button onClick={handleClick}>Click me</button>
  );
}
```

Conclusion

React permet de créer des applications web robustes et maintenables en utilisant des composants réutilisables, des hooks et une gestion efficace de l'état. Avec des outils comme React Router pour la navigation et Redux pour la gestion de l'état global, il est possible de développer des applications complexes de manière structurée et efficace.

Node JS

Node JS est un environnement d'exécution du langage JS hors du navigateur. Il est très utilisé dans le développement d'application en temps réel.

Grâce à Nodejs il devient alors possible d'embarquer le JS côté serveur, tout comme il est possible de le faire avec le PHP ou le Java ou PHP.

Installation

Pour installer le logiciel, il suffit d'exécuter le logiciel d'installation qu'on peut facilement trouver sur Internet aujourd'hui.

Utilisation

Nous avons principalement deux moyens de l'utiliser, soit en mode interactif ou bien nous enregistrons un fichier js que nous lançons avec nodejs.

Nous aurons ainsi la possibilité d'écrire nos scripts et les faire interpréter par le moteur d'exécution js.

Cas pratique: un serveur web

Nous allons mettre en pratique ces notions en créant un serveur web simple qui héberge une API de gestion des utilisateurs. Mais avant expliquons les notions sur les APIs.

Introduction aux API REST

Une API (Application Programming Interface) est un ensemble de règles qui permet à des applications de communiquer entre elles. REST (Representational State Transfer) est un style architectural pour concevoir des API.

Concepts Clés des API REST

Ressources

Définition : Une ressource est toute information qui peut être nommée, telle qu'un document ou une image. Identifiant : Chaque ressource est identifiée par un URI (Uniform Resource Identifier). Par exemple : https://api.exemple.com/utilisateurs/123.

Verbes HTTP

GET : Récupérer une ressource. **POST** : Créer une nouvelle ressource. **PUT** : Mettre à jour une ressource existante. **DELETE** : Supprimer une ressource. **PATCH** : Mettre à jour partiellement une ressource.

Statuts HTTP :

200 OK : La requête a réussi. **201 Created** : Une ressource a été créée. **204 No Content** : La requête a réussi mais ne retourne aucune ressource. **400 Bad Request** : La requête est invalide. **401 Unauthorized** : Authentification nécessaire. **404 Not Found** : Ressource non trouvée. **500 Internal Server Error** : Erreur côté serveur.

Exemples de Requêtes REST

- Récupérer tous les utilisateurs :

```
GET /utilisateurs
```

- Récupérer un utilisateur spécifique :

```
GET /utilisateurs/123
```

- Créer un nouvel utilisateur :

```
POST /utilisateurs
{
  "nom": "John Doe",
  "email": "john.doe@example.com"
}
```

- Mettre à jour un utilisateur existant :

```
PUT /utilisateurs/123
{
  "nom": "Jane Doe",
  "email": "jane.doe@example.com"
}
```

- Supprimer un utilisateur :

```
DELETE /utilisateurs/123
```

Bonnes Pratiques

- **Versionnement :** Inclure une version dans l'URL de l'API (par exemple, v1, v2).
- **Utilisation du JSON :** JSON est le format de choix pour les réponses et les requêtes car il est léger et largement utilisé.
- **Sécurité :** Utiliser HTTPS pour sécuriser les communications et les jetons d'authentification (comme OAuth).
- **Documentation :** Fournir une documentation claire et détaillée pour aider les développeurs à comprendre et utiliser l'API.

Conclusion

Les API REST sont essentielles pour la communication entre les applications dans un environnement distribué. En suivant les principes REST et les bonnes pratiques, on peut créer des API robustes, flexibles et faciles à maintenir