Partie 7
Frameworks JavaScript

Chapitre 7-1
Positionnement des frameworks JavaScript

1. Présentation générale des frameworks JavaScript

De très nombreux frameworks JavaScript existent, avec des positionnements fonctionnels différents.

Il ne peut être question, dans le cadre de ce livre réservé à des débutants en JavaScript, d'en faire une revue exhaustive.

Ils ont tous les points communs suivants : masquer la complexité du langage JavaScript, apporter de la robustesse dans les développements et aussi permettre, pour certains d'entre eux, d'interagir avec des bases de données.

1.1 Frameworks « front-end »

Les plus populaires des frameworks dits « front-end », c'est-à-dire gérant le côté interface utilisateur des applications web ou mobiles (téléphones mobiles, tablettes…) sont :

– Angular, framework développé par Google (la première version était connue sous l'appellation AngularJS)
– React JS (ou React), framework développé par Facebook
– Vue.js
– Svelte

1.2 Frameworks « back-end »

Pour les interactions avec les systèmes de gestion de bases de données, des frameworks dits « back-end » existent. Ils sont souvent eux-mêmes basés sur Node.js, qui est un environnement d'exécution multiplateforme Open Source exécutant du code JavaScript en dehors d'un navigateur (dans un runtime).

Node.js permet de concevoir des services d'accès à des Bases De Données et à des ressources disponibles sur Internet. Il fonctionne parfaitement sur Windows, Linux ou encore macOS.

Nous verrons par exemple que l'accès aux données pour le framework Svelte peut être assuré par les frameworks Express ou Sapper, tous deux basés sur Node.js.

1.3 Solutions de développement « hybride »

Pour terminer ce rapide panorama concernant les frameworks JavaScript, n'oublions pas les solutions dédiées aux développements pour périphériques mobiles (smartphones et tablettes). Dans un précédent livre, publié en 2020 aux Éditions ENI (Java et Ionic - Développement mobile pour Android : natif vs hybride), le framework Ionic a été présenté. Ce framework est lui-même basé sur d'autres frameworks, dont le très réputé Angular (soutenu par Google) et Apache Cordova.

Un chapitre du présent livre sera aussi consacré au Framework React Native (assez proche de React qui, lui, est réservé aux applications web). React Native permet très facilement à des développeurs ayant déjà une réelle expérience en React de concevoir des applications pour mobiles. Ce framework extrêmement utilisé et supporté par Facebook est une alternative plus que crédible à Ionic. Les applications React Native sont facilement déployables sur les mobiles fonctionnant sous systèmes d'exploitation Android et iOS (iPhone, iPad).

2. Les frameworks Node.js, Svelte, React et React Native

Comme indiqué précédemment, un court chapitre (Installation de Node.js) sera consacré à l'installation du framework Node.js, socle sur lequel fonctionnent les frameworks Svelte, React et React Native.

Le framework Svelte, relativement récent, est un challenger crédible pour React (React JS) et Vue.js. Svelte, présenté dans le chapitre Framework Svelte, possède de nombreux atouts techniques. Il bénéficie par contre pour l'instant d'une communauté de développeurs plus restreinte que celles des deux acteurs principaux (React et Vue.js).

Dans le cadre de ce livre, un choix éditorial a été fait de ne pas évoquer Vue.js. Vous trouvez, toujours aux Éditions ENI, des ouvrages dédiés exclusivement à Vue.js, notamment le livre Vue.js - Développez des applications web modernes en JavaScript de Yoann GAUCHARD.

Le chapitre Framework React sera celui dédié à React. Comme dans le chapitre consacré à Svelte, après une rapide présentation des concepts de base, de nombreux exemples seront proposés et largement commentés.

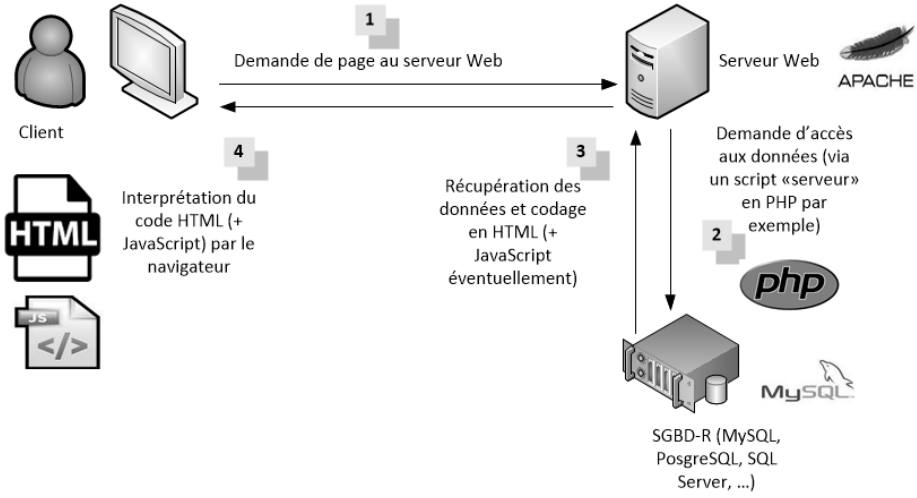
Le livre se terminera au chapitre Framework React Native avec un exposé consacré à React Native, la version du framework React permettant le développement d'applications pour mobiles.

Chapitre 7-2
Installation de Node.js

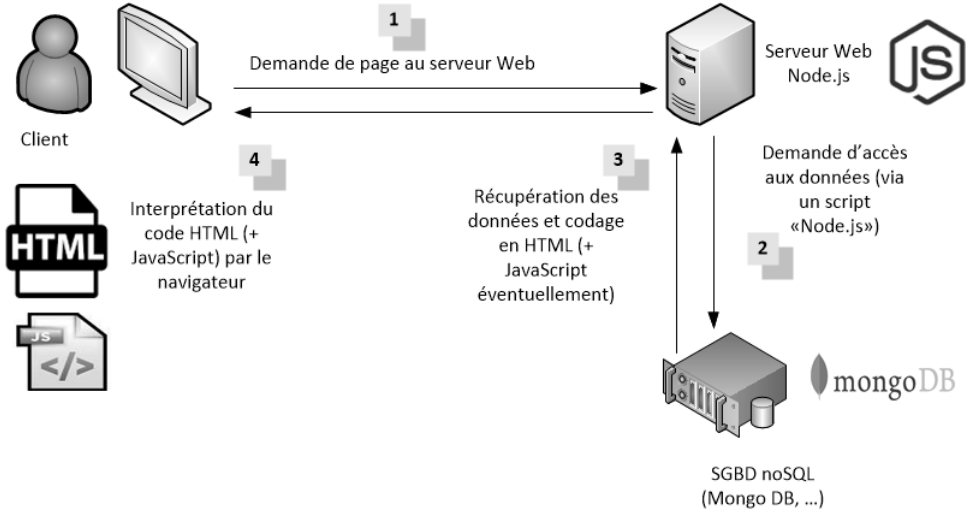
1. Présentation du framework Node.js

JavaScript a longtemps été cantonné à une utilisation côté client. On utilisait JavaScript seulement pour ajouter de l'interactivité dans les pages web (animations, contrôles de saisie...).

Pour les accès aux Bases De Données distantes, comme MySQL, les applications web intégraient par ailleurs des scripts orientés serveur, là aussi souvent développés en langage PHP.



Avec Node.js, il est bien sûr toujours possible d'utiliser JavaScript côté client pour manipuler les pages HTML. En plus, Node.js propose un environnement côté serveur qui permet aussi d'utiliser le langage JavaScript pour générer des pages web. En clair, il vient en remplacement de langages serveur comme PHP, Java EE, etc.



Partie 2 : L'HTML 5.2

Chapitre 2-1
Les éléments HTML

1. Bien utiliser le HTML

Le langage HTML est un langage à balises, comme son nom l'indique : **HyperText Markup Language**, Langage à Balises Hypertextes. Notez aussi que le HTML est originellement une application SGML (*Standard Generalized Markup Language*).

Le langage HTML est saisi dans un document de type texte, ayant pour extension **.html**. Le HTML est interprété par un « agent utilisateur » pour reprendre la terminologie officielle (*user agent*). Ces agents utilisateurs du HTML sont la plupart du temps des navigateurs web, mais il existe d'autres formes d'applications capables d'interpréter le HTML, comme les lecteurs d'écran audio, les robots d'indexation des moteurs de recherche, mais aussi des systèmes embarqués dans des appareils électroniques.

L'objectif du HTML est de décrire la structure des pages web et d'indiquer le contenu sémantique de chaque élément constitutif de ces pages. Le HTML décrit le contenu des pages web à l'aide d'éléments HTML qui sont constitués de balises. Chaque élément est dédié à l'affichage d'un type de contenu donné. Vous avez des éléments HTML qui affichent les titres, les images, les tableaux, les formulaires…

Le HTML est donc bien un **langage sémantique** : chaque élément doit être utilisé dans le cadre de sa définition et les agents utilisateurs attendent un contenu donné pour chaque élément. Par exemple, l'élément HTML est fait pour contenir un paragraphe de texte et non un tableau, l'élément HTML <img> est fait pour contenir une image et non un champ de formulaire. Pour que vos pages web soient valides et bien interprétées par les navigateurs, vous devez respecter cette sémantique.

2. Les balises et les contenus

Chaque élément HTML est composé de plusieurs parties constitutives.

La première partie est la balise d'ouverture. Cette balise commence par le caractère < et est immédiatement suivi par le nom de l'élément et se termine par le caractère >. L'élément qui permet d'insérer un paragraphe de texte est nommé p, la syntaxe de sa balise d'ouverture est la suivante : .

La deuxième partie concerne les contenus rédactionnels. C'est-à-dire les éléments HTML qui contiennent du texte, comme les titres, les paragraphes, les citations… Le texte est alors écrit de manière usuelle, sans aucune balise spécifique.

La plupart des éléments HTML ont une balise de fermeture. La syntaxe reprend le principe de la balise d'ouverture, mais avec en plus le caractère / qui précède le nom de l'élément. Ce caractère indique la fin de l'élément. Si nous reprenons notre exemple de paragraphe, voici la balise de fermeture : .

Voici un exemple complet pour l'élément HTML insérant un paragraphe :

```
<p>Le texte de mon paragraphe.</p>
```

Il est important de noter que certains éléments HTML n'ayant pas de contenu rédactionnel ou pas d'éléments imbriqués n'ont logiquement pas de balise de fermeture. Prenons pour exemple l'élément <hr> qui permet d'insérer une ligne horizontale de séparation. Il n'y a pas de contenu rédactionnel, donc nous n'avons pas de balise de fermeture. Il en est de même avec l'élément <img> qui permet d'insérer une image. Dans ce cas, la balise d'ouverture est qualifiée d'autofermante.

3. Les attributs des éléments

Les attributs permettent de modifier le comportement standard des éléments HTML. Les éléments HTML peuvent ne contenir aucun attribut, ou un seul et parfois plusieurs. Certains attributs sont obligatoires, d'autres sont facultatifs. La plupart des attributs ont des valeurs, mais pas tous. Les attributs qui ne nécessitent pas de valeur sont qualifiés de booléens. Dans tous les cas, les attributs se placent dans la balise d'ouverture des éléments.

Prenons comme premier exemple l'attribut facultatif qui permet d'identifier de manière unique un élément HTML. C'est l'attribut `id`. Cet attribut doit avoir une valeur qui est indiquée entre guillemets (non obligatoires, mais très fortement conseillés) et qui est précédée par le signe =. Voici un exemple :

```
<p id="introduction">Le contenu de mon paragraphe.</p>
```

Prenons comme deuxième exemple un attribut obligatoire. Pour insérer une image, nous utilisons l'élément `<img>` et son attribut obligatoire `src` qui permet d'indiquer le chemin d'accès à la source de l'image. Voici une syntaxe simple :

```
<img src="mon-image.jpg">
```

Si un élément HTML possède plusieurs attributs, ils sont séparés par un simple espace. Voici un exemple avec de nouveau l'élément `<img>` :

```
<img src="mon-image.jpg" alt="Le texte alternatif de l'image"
title="Titre de l'image">
```

Notez que chaque élément HTML peut avoir des attributs qui lui sont propres. Mais sachez qu'il existe de très nombreux attributs universels (*Global attributes* en anglais) qui peuvent s'appliquer à tous les éléments HTML. Voici une URL où vous pourrez trouver la liste de ces attributs universels :
https://developer.mozilla.org/fr/docs/Web/HTML/Attributs_universels

Voici quelques attributs couramment utilisés :

– `accesskey` : indique la touche clavier permettant d'accéder à un élément de l'interface. Exemple : `accesskey="A"`.
– `class` : spécifie le nom de la classe CSS à appliquer à l'élément. Exemple : `class="titre-intro"`.

– `contenteditable` : booléen indiquant si le contenu est éditable ou non par l'utilisateur. Exemple : `contenteditable="true"`.
– `dir` : indique la direction de l'écriture, de gauche à droite ou de droite à gauche. Exemple : `dir="ltr"`, pour *Left To Right*.
– `hidden` : booléen indiquant si l'élément est masqué ou non. Exemple : `hidden`, la présence même de cet attribut suffit pour indiquer la valeur `true`.
– `id` : détermine l'identifiant unique de l'élément.
Exemple : `id="banniere-article"`.
– `lang` : indique la langue utilisée dans l'élément. Exemple : `lang="en"`.
– `title` : donne le titre descriptif de l'élément.
Exemple : `title="Hypertext Markup Language"`.
– `style` : applique des propriétés de style CSS à l'élément.
Exemple : `style="color: #00FF00 ; text-align: center"`.

4. Le bon usage de la syntaxe

Le langage HTML est un langage assez permissif, mais il convient de respecter certaines règles pour offrir aux différents intervenants un code propre, lisible et valide.

Vous pouvez parfaitement utiliser des majuscules ou des minuscules pour saisir le nom des éléments et des attributs HTML.
Les syntaxes `<p>Mon texte.</p>` et `<P>Mon texte.</P>` sont équivalentes. Mais l'usage prévaut l'utilisation des minuscules.

Certains éléments HTML ont une balise de fermeture optionnelle, comme l'élément `<p>`. Mais pour les mêmes raisons que précédemment, fermer toujours les éléments de contenu par leur balise de fermeture.

Les valeurs des attributs peuvent ne pas être indiquées entre guillemets. Mais à nouveau, privilégiez toujours les guillemets.

5. L'imbrication des éléments

Les éléments HTML permettent de structurer le contenu de vos pages web. Cette structuration va de pair avec l'imbrication des éléments HTML. Par exemple dans un article, inséré avec l'élément `<article>`, nous pouvons imbriquer un élément d'en-tête `<header>`, un ou plusieurs paragraphes `<p>` et un pied de page `<footer>`. Nous avons alors une hiérarchie des éléments : `<header>`, `<p>` et `<footer>` sont imbriqués dans `<article>`. Ces éléments sont les enfants de l'article qui est leur parent. S'il y a plusieurs enfants `<p>`, ils sont frères.

Dans les paragraphes `<p>`, nous pouvons parfaitement appliquer une mise en forme sémantique avec l'élément `<strong>` qui permet d'appliquer une forte emphase. Dans ce cas, l'élément `<strong>` est enfant de l'élément `<p>`.

Avec ces exemples, nous voyons, en simplifiant, deux types d'éléments HTML. Les éléments de structure et les éléments de mise en forme du texte. Cette notion est héritée du HTML 4. Dans cette recommandation, les éléments HTML étaient typés en "*block*" et en "*inline*". Les éléments de type bloc (*block*) permettent de structurer la page, avec des éléments comme `<div>`, `<p>`, `<h1>`. Par défaut, les navigateurs doivent afficher ces éléments sur toute la largeur disponible et ils doivent commencer sur une nouvelle ligne. Les éléments en ligne (*inline*) permettent de mettre en forme le texte. Les navigateurs doivent donc les afficher dans la même ligne et ils peuvent s'enchaîner les uns aux autres. Bien sûr, ces deux types sont les plus utilisés et il en existe bien d'autres. Voici une URL qui vous liste tous les types d'affichage : https://www.w3schools.com/cssref/pr_class_display.asp.

Donc, en ce qui concerne l'imbrication des éléments, les éléments de type bloc peuvent contenir d'autres éléments de type bloc, des éléments de type en ligne et du texte. Les éléments de type en ligne peuvent contenir d'autres éléments en ligne ou du texte, mais pas d'éléments de type bloc.

Prenons un exemple très simple. Nous devons saisir un texte dont certains mots doivent être mis en évidence. Nous allons donc utiliser un élément `<p>` de type bloc, contenant du texte, dont un mot sera mis en évidence avec l'élément `<strong>` de type en ligne.

```
<p>Mon texte est mis en <strong>évidence</strong>
avec une emphase forte.</p>
```

L'imbrication inverse n'est pas possible, c'est une évidence :

```
<strong>Mon texte est mis en <p>évidence</p>
avec une emphase forte.</strong>
```

Ces notions de type *block* (https://developer.mozilla.org/fr/docs/Web/HTML/Éléments_en_bloc) et *inline* (https://developer.mozilla.org/fr/docs/Web/HTML/Éléments_en_ligne), et les autres types étaient parfaitement bien définis dans le HTML 4. Mais avec le HTML5 ces différences se sont un peu estompées. Par exemple, il est autorisé d'avoir un élément enfant `<p>`, de type bloc, dans un élément parent `<a>`, de type en ligne.

6. Les commentaires

Comme dans tout langage informatique, il est vivement conseillé de commenter son code, que ce soit pour vous ou pour un autre développeur qui viendra reprendre vos pages. Les commentaires peuvent se placer n'importe où dans la page. Voici la syntaxe :

```
<!-- Le texte de mon commentaire -->
```