Architecture(s) et application(s) Web

Télécom SudParis

24/08/2023

CSC4101 - Expérience utilisateur Web

Table des matières

Objectifs de cette séquence

Cette quatrième séquence de cours porte sur différents aspects de l'**expérience utilisateur** dans l'interaction avec les applications Web.
On présentera les enjeux généraux de l'**ergonomie** (autrement appelée « expérience utilisateur ») des interfaces utilisateur ainsi que les problématiques d'**accessibilité**.
On détaillera les fonctionnalités principales, dans cette optique, pour la construction et l'**habillage des pages HTML**, que sont les standards ***Document Object Model*** (DOM) et ***Cascading Style Sheets*** (CSS).
On verra enfin comment les contrôleurs de l'application Symfony permettent de gérer les transitions hypertexte au sein des pages de l'interface d'une application Web.

1 Interface utilisateur

Cette section présente les principes généraux qui guident la conception d'interface utilisateur, en général (y compris en dehors du domaine des applications Web)

1.1 Interface homme-machine (IHM)

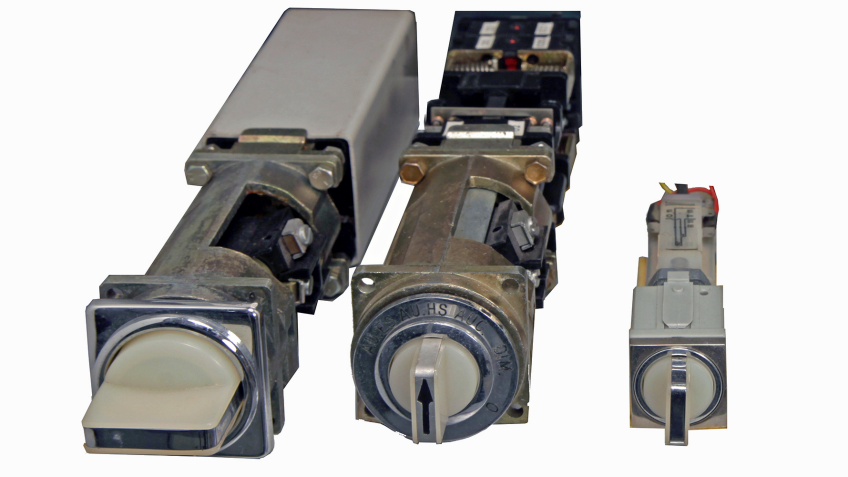


Figure 1 – Boutons Tournés Poussés Lumineux

Cette illustration présente des boutons « Tournés Poussés Lumineux ». Leur fonction est d'éviter la commande d'organes sensibles : typiquement commande d'organes dans des postes de transformation électrique ou centrales nucléaires. Le but est de forcer l'opérateur à réfléchir à deux fois :

1. tourner : choisir la fonction
2. pousser : activer la fonction
3. allumage lampe : acquittement de la fonction correctement exécutée

Ce genre de widget est difficile à informatiser.
Pour la petite histoire on travaillait sur ce genre de fonctionnalité pour EDF dans les années 90, pour les postes d'exploitation informatisés : GUI sous X/Motif au lieu de tableau synoptiques physiques…

1.2 Interface utilisateur

Vous connaissez bien cette interface utilisateur, en ligne de commande.

1.3 Propriétés de la ligne de commande

Command-Line Interface (CLI) :
- Codifiée
- Stricte
- Statique

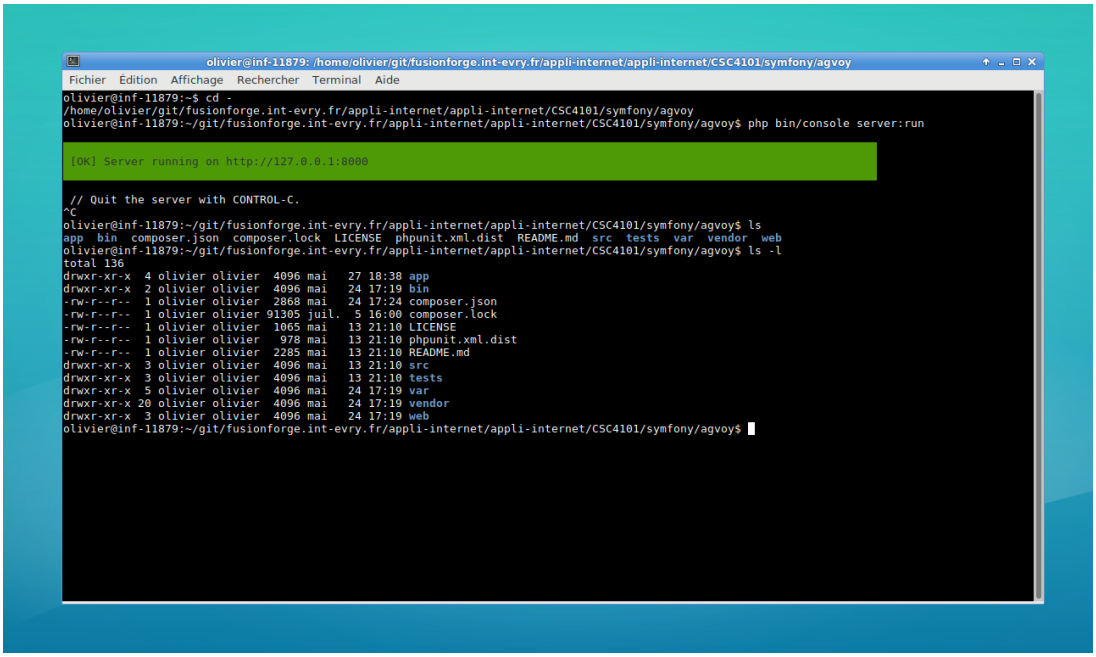


Figure 2 – Ligne de commande bash

Cf. CSC3102
Améliorations possibles avec libcurses, etc.
Pratique : tunnel via SSH : peu de bande passante, *screen* pour rendre la session rémanente, Emacs, convient bien aux interfaces braille, etc.

1.4 Interface utilisateur

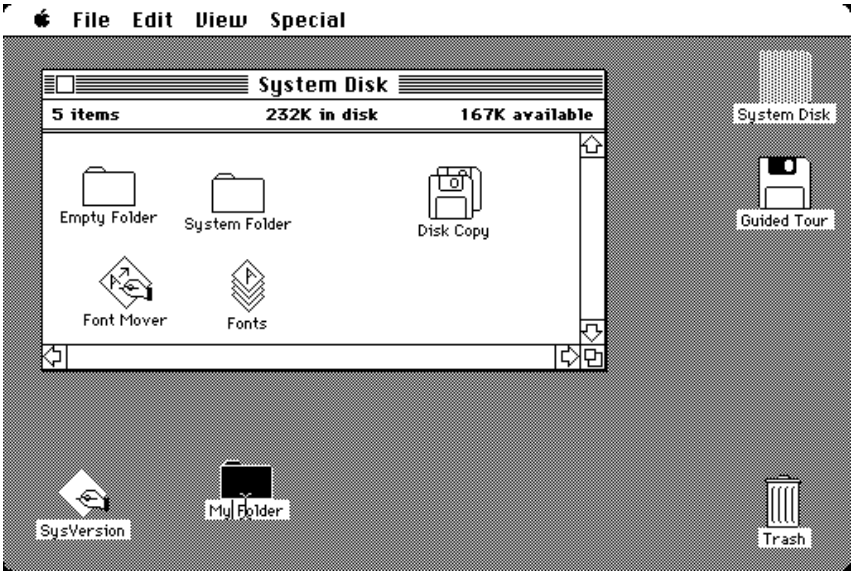


Figure 3 – Desktop du Macintosh d'Apple

Un peu avant le Macintosh d'Apple, le Xerox Alto invente l'interface graphique avec souris.
Cf. Xerox PARC et la naissance de l'informatique contemporaine par Sacha Krakowiak
Anecdote perso : j'ai utilisé une telle interface sur Geos sur Commodore 64 pour mon premier rapport rendu en *WYSIWYG* (*What You See Is What You Get*) au collège dans les années 80.

1.5 Propriétés des interfaces graphiques

Graphical User Interface (GUI) :
- Métaphore (bureau)
- Exploratoire

1.6 Interfaces de 0 à 77 ans

Vidéo `https://www.youtube.com/watch?v=gc9NpYrbZgQ` par `UserExperiencesWorks`

1.7 Propriétés des interfaces « naturelles »

Natural User Interface (NUI) :
- Directe (passage de novice à expert facilement)
- Naturelle / Intuitive

1.8 Importance des ordiphones (mobiles)

Plate-forme préférentielle (majorité des utilisateurs)
Utiliser technos Web pour applis mobiles :-)

2 Interfaces Web

Voyons les caractéristiques propres des interfaces des applications Web.

2.1 Qualité des interfaces Web

On va passer en revue quelques grandes propriétés intéressantes du point de vue de la qualité des interfaces des applications Web.

2.1.1 Ergonomie

- **Expérience utilisateur** (*User eXperience : UX*)
- Utilisabilité :
  - apprenabilité (novices)
- « Fléau » de l'abandon de panier
- Utilisateurs « PIP » : *Pressés, Ignorants et Paresseux*

Le but ultime est que l'« expérience » de navigation sur l'application soit bonne pour les utilisateurs… sachant qu'ils n'ont pas tous les mêmes compétences, ni les mêmes capacités.
C'est un art difficile à atteindre, en pratique.

2.1.2 Portabilité

Le Web est la plate-forme universelle.

- Standardisation = portabilité (merci HTML5)
- Applications mobiles :
  - Développé en HTML
  - « Compilé » en *toolkit* natif (ex. Apache Cordova)
- **Attention** : prédominence de Chrome de Google… *Best viewed in Chrome*

Sur le contexte de la standardisation, et l'émergence actuelle de Chrome, les « *Browser Wars* » : `https://fr.wikipedia.org/wiki/%C3%89volution_de_l'usage_des_navigateurs_web`

2.2 Accessibilité

Vidéo : `https://www.youtube.com/watch?v=DePdWynmd_Y`
Source : article Comment les aveugles utilisent-ils internet ? de *l'Obs* Voir aussi Usage plage braille et synthèse vocale par personne non voyante de `https://design.numerique.gouv.fr/`.

Cette vidéo illustre comment les aveugles ou mal-voyants utilisent le Web, avec plage braille et logiciel de synthèse vocale.
La problématique de l'accessibilité dépasse de loin la situation des personnes en situation de handicap « reconnue ». Elle concerne également tous les utilisateurs à partir du moment où leur utilisation des applications Web n'est pas idéale, car ils sont en mode « dégradé » (petit écran sur *smartphone*, un seul doigt de libre, réseau lent, environnement bruyant, etc.)

2.2.1 Obligations

- Règles pour l'accessibilité des contenus Web (WCAG) 2.0
  - Différents niveaux de priorité / niveaux de conformité à la norme
  - Principes : Perceptible, Utilisable, Compréhensible, Robuste
  - Règles, critères de succès
  - Techniques suffisantes et techniques recommandées
- Référentiel Général d'Accessibilité des Administrations (RGAA) version 4.1.2 (avril 2023)

Certains éléments du standard *Web Content Accessibility Guidelines* (WCAG) deviennent obligatoires, par exemple pour les sites Web des administrations publiques, d'où le RGAA.
Référence sur le sujet : `http://www.accessiweb.org/` de `http://www.braillenet.org/`

2.2.2 Principes WCAG 2.0

2.2.3 Principe 1 : perceptible

- 1.1 Proposer des **équivalents textuels** à tout contenu non textuel qui pourra alors être présenté sous d'autres formes selon les besoins de l'utilisateur : grands caractères, braille, synthèse vocale, symboles ou langage simplifié.
- 1.2 Proposer des versions de remplacement aux média temporels.
- 1.3 Créer un contenu qui puisse être présenté de différentes manières sans perte d'information ni de structure (par exemple avec une mise en page simplifiée).
- 1.4 Faciliter la perception visuelle et auditive du contenu par l'utilisateur, notamment en séparant le premier plan de l'arrière-plan.

2.2.4 Principe 2 : utilisable

- 2.1 Rendre toutes les fonctionnalités **accessibles au clavier**.
- 2.2 Laisser à l'utilisateur suffisamment de temps pour lire et utiliser le contenu.
- 2.3 Ne pas concevoir de contenu susceptible de **provoquer des crises**.
- 2.4 Fournir à l'utilisateur des **éléments d'orientation** pour naviguer, trouver le contenu et se situer dans le site.

2.2.5 Principe 3 : compréhensible

- 3.1 Rendre le contenu textuel lisible et compréhensible.
- 3.2 Faire en sorte que les pages apparaissent et fonctionnent **de manière prévisible**.
- 3.3 Aider l'utilisateur à éviter et à corriger les erreurs de saisie.

2.2.6 Principe 4 : robuste

- 4.1 Optimiser la compatibilité avec les agents utilisateurs actuels et futurs, y compris avec les technologies d'assistance.

etc.

2.2.7 La suite sur DesignGouv

« Concevons des services publics numériques accessibles, inclusifs et humains. » : `https://design.numerique.gouv.fr/`

2.2.8 Évaluation qualité

- Critères Opquast (*Open Quality Standards*) : `https://checklists.opquast.com/fr/qualiteweb/`

Cf. bibliographie en annexes

3 Habillage des pages Web

Voyons maintenant ce qui concerne la mise en forme, l'habillage des pages Web de l'application.

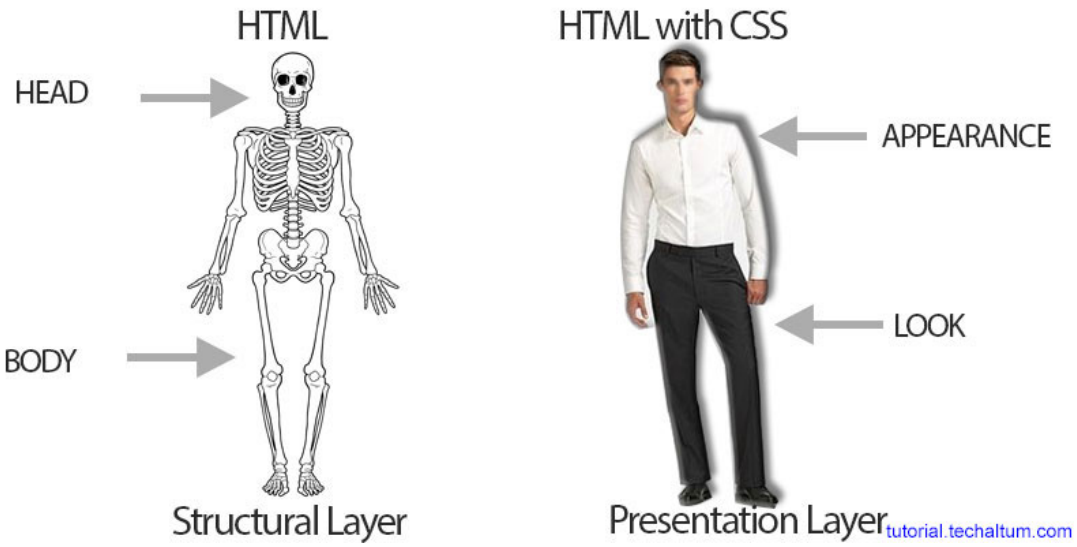
3.1 Découplage structure / présentation

(*suite*)

3.1.1 Page Web et document HTML

Conversion de **documents** en **une présentation** dans le navigateur.

3.1.2 HTML + CSS



Chaque langage a son rôle :
- HTML (5) : la structure du document, essentiellement sémantique
- CSS : les règles de présentation

3.1.3 Structure des sources des pages

Une « page Web » :
- un document HTML (maître)
- plus des images, boutons, menus, etc. (éventuellement externes)

Un document (HTML) :
- un arbre de rubriques,
- sous-rubriques,
- etc.

Arbre de balises HTML (DOM : *Document Object Model*)

On visualise des ressources (URL), dont le serveur produit une représentation en HTML, pour qu'elle soit visualisée par un navigateur.
La structure d'un document HTML est essentiellement arborescente (langage à balises ouvrantes/fermantes).
En général on associe une sémantique à cette arborescence : décomposition en concepts, sous-concepts, etc.
Un standard existe pour décrire et parcourir cet arbre : DOM (*Document Object Model*).
Mais l'utilisateur humain ne consulte pas un arbre de balises.

3.1.4 Structure d'une page affichée

Une page affichée en 2D (ou imprimée) :
- des boîtes qui contiennent d'autres boîtes
- boîtes qui contiennent texte ou images, etc.
- texte qui contient des caractères
- caractère qui contiennent des pixels
- …

C'est plus ou moins le modèle d'une page d'un document papier.
Là, la page est purement syntaxique (on perd la structure sémantique du document).
On trouve ce genre de structure dans PostScript par exemple.

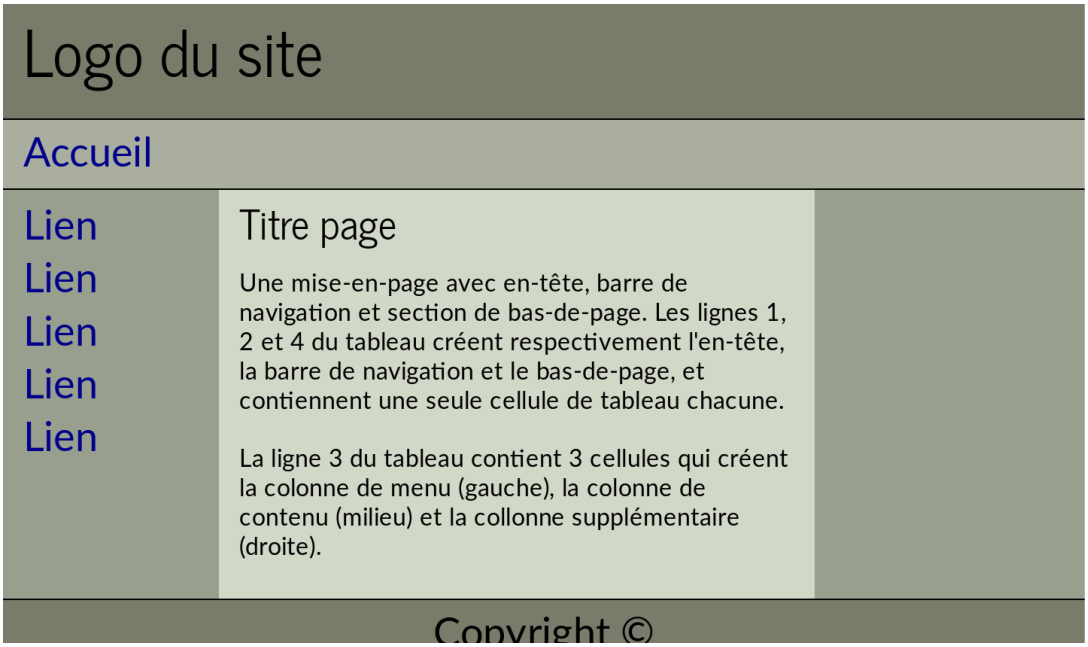
3.1.5 Arbre DOM et rendu des pages

- Le navigateur (moteur de rendu) convertit :
  - Arbre DOM : *Document Object Model*

  en :
  - Arbre d'éléments à afficher
- **Règles de conversion**
  - Prédéfinies (navigateur)
  - Programmables (**CSS**)

Le travail du navigateur est la conversion de l'arbre de balises en une page à deux dimensions équivalente au texte d'un document papier.
Le moteur de rendu du navigateur est un engin loin d'être trivial.
Le programmeur n'a pas besoin de spécifier la façon dont ce rendu est effectué, mais il peut le faire en détaillant les attributs de mise en forme des éléments graphiques (CSS).

3.1.6 Exemple de page avec un tableau



On décide d'afficher à l'utilisateur une page Web structurée globalement comme un tableau.
Toutes les cases de ce tableau n'ont pas la même taille, la même couleur, etc.
Comment peut-on construire un document HTML pour représenter cette page.

3.1.7 Structure HTML correspondante ?

```
<table>
  <tr> <!-- SECTION EN-TETE  -->
    <td colspan="3">
      <h3>Logo du site</h3>
    </td>
  </tr>
  <tr>  <!--  SECTION BARRE NAVIGATION == -->
    <td colspan="3">
      <a href="#">Accueil</a>
    </td>
  </tr>
  <tr>
    <td width="20%"> <!--  COLONNE GAUCHE (MENU) == -->
      <a href="#">Lien</a><br>
      <a href="#">Lien</a><br>
    </td>
    <td width="55%"> <!--  COLONNE MILIEU (CONTENU) == -->
      <h4>Titre page</h4>
      Une mise-en-page avec en-tête, barre de navigation et section de
      ...
    </td>
    <td width="25%"></td>
  </tr>
  <tr>  <!--  SECTION PIED-DE-PAGE == -->
    <td colspan="3">Copyright ©</td>
  </tr>
</table>
```

Globalement, cette approche fonctionne… mais c'est très verbeux, et un peu de la bidouille.
Les balises HTML des différentes cellules se ressemblent toutes. Difficile de s'y repérer.
Dur aussi de faire évoluer la mise en page (par exemple agrandir ou diminuer la taille des colonnes, ou ajouter des sous-cases.

3.1.8 Faire mieux !

- Structurer la page en 6 **sections**, par exemple, avec des balises `<div>`
- Positionner le contenu de chaque section avec des règles CSS.
- Permet de faire évoluer la mise en page, par exemple sous forme linéaire et non plus tabulaire, etc.

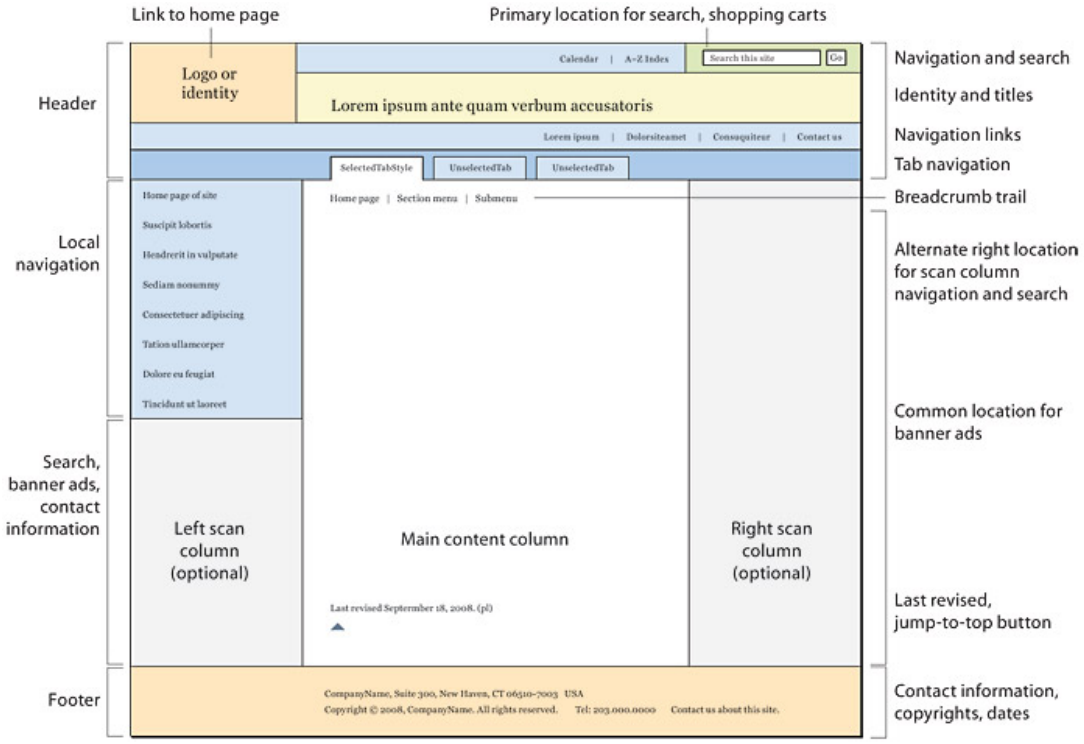
On peut donc donner une structuration sémantique aux sections :
- en-tête
- menu
- cœur du texte,
- …

Le moteur de rendu doit alors être programmé de façon précise pour positionner les sections :
- menu : à gauche
- cœur du texte : 60 % de la largeur
- etc.

C'est le rôle de CSS de faire se positionnement, sans polluer le texte du source HTML avec toutes ces directives.
C'est beaucoup plus évolutif, et adaptable (rendu sur petit écran, etc.)

3.2 Programmer en HTML et CSS

3.2.1 Structure archétypale d'une page



Source : `http://webstyleguide.com/wsg3/6-page-structure/3-site-design.html`

Certains éléments se répètent sur tout le site
Seul le contenu de la partie centrale est réellement spécifique à chaque page
Nécessite de factoriser le code, les éléments de construction des pages, etc.

3.2.2 Exemples : *CSS Zen Garden*

`http://www.csszengarden.com/tr/francais/`

Même page HTML (voir sans feuille de style dans navigateur et consulter source HTML), mais différentes feuilles de style.
C'est tout l'intérêt du CSS : changer le look d'un site / application sans avoir à retoucher le code HTML ou les programmes, mais uniquement le CSS.

3.2.3 Design

Outils de *Mockup* :
- Papier + crayon + gomme
- Outils (penpot)
- HTML + CSS (+ *templates*)

En français : maquettage.
Métier à part entière : *Web design*, avec des agences spécialisées

3.2.4 Guides de style

- Chercher « *web interface style guide* » sur votre moteur de recherche préféré
- Faire comme sur les applis mobiles :
  - Android : *Material design* : `https://developer.android.com/design/index.html`
  - Apple : *Designing for iOS* : `https://developer.apple.com/design/human-interface-guidelines/designing-for-ios`

Des goûts et des couleurs ?
Attention à respecter un certain nombre de règles pour ne pas dérouter les utilisateurs. Mieux vaut s'inspirer de ce qu'ils connaissent.
Dans cette séquence, nous n'allons pas approfondir les différents principes d'ergonomie et les variantes, mais vous pourrez approfondir en travail personnel.

3.2.5 Inspiration / prêt à l'emploi

- `https://www.awwwards.com/` : récompenses pour le *Web design*
- `http://www.webdesign-inspiration.com/` : sélection de *designs*

Marché :
- par ex. : `https://themeforest.net/` : thèmes prêts à l'emploi
- *plein d'autres*

3.3 Caractéristiques de HTML

3.3.1 HTML 5

- Tout (ou presque) est possible en terme d'interface d'applications.
- Y compris l'accès à des zones de graphisme libre et l'intégration des périphériques du téléphone/ordinateur

Exemples :
- ~~http://trumpdonald.org/~~ `https://funkypotato.com/gamez/trump-donald/index.html` (HTML5 « natif » ?)
- `http://funhtml5games.com/?play=angrybirds` (portage de GWT -> HTML5)
- `https://princejs.com/`

3.3.2 Les bons artistes copient, les grands artistes volent

Avantage d'HTML : le code source HTML dispo.
Vive le copier-coller :-)

Citation attribuée à Pablo Picasso
C'est ce qui a assuré le succès de HTML.

3.3.3 Contenu adaptatif

- *Responsive design* : prend en compte la taille de l'écran automatiquement
- p. ex. : avec Bootstrap (voir plus loin)

Principe : masquer ou faire varier les propriétés des éléments de la page pour visualiser plus ou moins de détails.

3.3.4 Interactions dynamiques

- Javascript
- Interactions asynchrones

Javascript va pouvoir faire évoluer la représentation du document obtenue dans la page Web du navigateur, sans avoir à refaire une requête auprès du serveur.
Exemple : naviguer dans différents « onglets » : les onglets sont tous obtenus lors de la même requête HTTP et contenus dans différentes sous-sections du même document HTML. Mais un seul onglet est visible dans une zone de la page à un instant donné.
À l'inverse, il est possible de ne charger qu'un document maître partiel, qui est affiché initialement, et peupler ensuite le contenu des sous-rubriques de ce document en tâche de fond ou à la demande, en complétant avec le résultat d'autres requêtes.
C'est le principe d'AJAX (cf. séquence ultérieure).

3.3.5 Interactions avec ordiphone mobile

- Application allant plus loin que l'affichage et la saisie de texte : accès à toutes les fonctions du terminal mobile depuis le navigateur Web

Roadmap of Web Applications on Mobile du W3C (September 2020)

Applications à base de technologies Web, mais utilisant les dispositifs autres qu'écran, clavier, souris et haut-parleur :
- micro
- webcam
- gyroscopes
- GPS
- multi-touch
- etc.

4 Principes de CSS

Cascading Style Sheets (Feuilles de style de Cascade)
Concevoir l'habillage des pages du site / application

4.1 Cascade

Modèle « objet » particulier
- Langage déclaratif
- Combinaison de plusieurs feuilles avec priorités
- « Héritage » de propriétés des parents
- Factorisation de motifs :
  DRY – *don't repeat yourself*

4.2 Langage à base de règles

- **Si** motif trouvé (sélecteur)
- **alors** valeur donnée à attribut/propriété de mise en forme

4.3 Exemple

```
h1 {
  font-size: 60px;
  text-align: center;
}

p,
li {
  font-size: 16px;
  line-height: 2;
  letter-spacing: 1px;
}
```

4.4 Niveaux croissants de proximité

Emplacement du code CSS :

1. pas de style : affichage navigateur par défaut
2. fichier CSS externe :

   ```
   <link rel="stylesheet" href="external.css"/>
   ```

3. bloc style interne :

   ```
   <style type="text/css">
   .underline { text-decoration: underline; }
   </style>
   ```

4. dans élément :

   ```
   <div class="column" style="float:left; width: 50%">
   ```

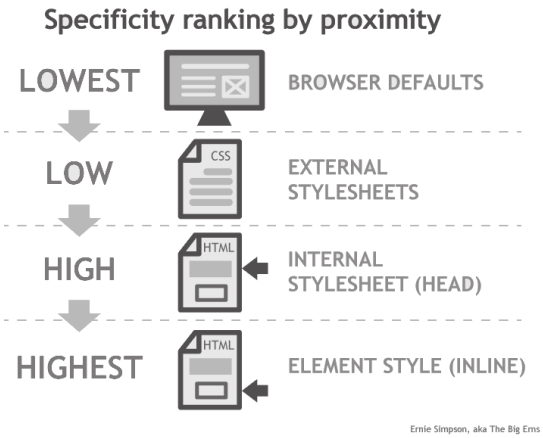


Figure 4 – Specificity ranking by proximity

Les illustrations de ces sections sont tirées de l'article ~~http://cssway.thebigerns.com/special/master-item-styles/~~ « Freeway Pro 6, Master Item styles and the Cascade Order » de Ernie Simpson
Ici, on entend par proximité, celle entre une définition d'une règle de rendu CSS, et un élément concerné, de l'arbre HTML.
Plus la spécification d'une règle CSS est proche, plus elle annule et remplace des règles plus éloignées qui s'appliqueraient à ce même élément.

4.5 Spécificité selon la proximité

Quelles règles s'appliquent, si plusieurs définies ?

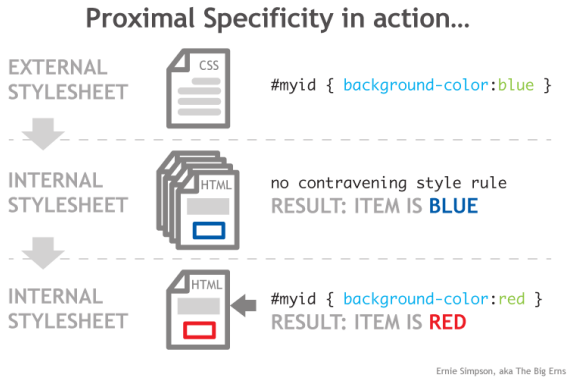


Figure 5 – Proximal specificity in action…

Dans cet exemple, une page spécialise, localement, un comportement défini pour l'ensemble du site.
On peut définir la couleur de fond bleu pour l'élement d'identifiant `myid`, via une feuille de style `external.css` contenant :

```
#myid { background-color: blue; }
```

Elle sera incluse dans le source HTML des pages du site :

```
<head>
  <link href="external.css" rel="stylesheet" type="text/css">
</head>
```

Dans toutes les pages qui contiennent cette feuille de style, un élément d'identifiant `myid` sera sur fond bleu.
Mais via une règle placée dans une page particulière, on peut surcharger cette règle, pour utiliser plutôt du rouge :

```
<head>
 <style type="text/css">
   #myid { background-color: red; }
 </style>
```

Encore plus près, on pourrait faire :

```
<p style="background-color: fuchsia;">Aren't style sheets wonderful?</p>
```

4.6 Spécificité du sélecteur

Ordre croissant de spécificité du sélecteur :

1. par balise : `div {style:valeur}`
2. par classe : `.myclass {style:valeur}`
3. par identifiant : `#myid {style:valeur}`

Figure 6 – Specificity ranking by selector type

Un sélecteur générique, comme le nom d'une balise (`<div>`) couvre beaucoup de contenu, et est donc moins spécifique qu'une classe (`.myclass`), ou bien qu'un identifiant unique (`#myid`).
Tout le jeu consistera à définir des règles générales, par exemple au niveau des classes, et à autoriser parfois des spécificités, pour certains identifiants.

4.7 Outils du développeur

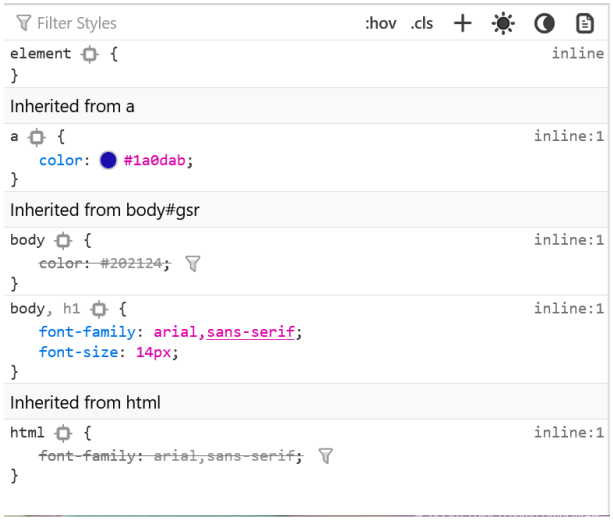


Figure 7 – Outil du développeur : règles CSS, dans Firefox

Source : Examiner et modifier le CSS sur MDN

4.8 Conception de CSS

- Pas toujours simple
- Abstraction
- *Frameworks* CSS (et Javascript)
  - ex. : Bootstrap

Couplage blocs *templates* Twig et CSS

Ne pas partir de rien et réutiliser des *frameworks*.
Les détails ne seront pas présentés en cours : travail hors présentiel.

5 Conception blocs Twig et CSS - Bootstrap

Examinons finalement des éléments méthodologiques et des outils qu'on va mettre en œuvre dans les projets.

5.1 Conception des gabarits et CSS

On peut définir des règles de structuration des pages HTML, au niveau des gabarits, en parallèle de la définition des règles de mise en forme.
Voici une méthodologie assez simple pour ce faire.

5.1.1 Concevoir les gabarits des pages

Conception parallèle :
- des blocs des gabarits HTML
- des sections `<div>` qu'ils contiennent (identifiants ou classes)
- des mises en forme

Même convention de nommage

5.1.2 Modèle de page souhaité

```
<html>
  <head>
  </head>
  <body>
    NAVIGATION

    MENU (COLONNE DE GAUCHE)

    CONTENU PRINCIPAL

    BAS DE PAGE
  </body>
</html>
```

5.1.3 Structure sémantique

```
<body>
  <div id="navigation">
    (NAVIGATION)
  </div>
  <div id="menu">
    (MENU RUBRIQUES)
  </div>
  <div id="main">
    (CONTENU PRINCIPAL)
  </div>
  <div id="footer">
    (BAS DE PAGE)
  </div>
</body>
```

Pour chaque grande zone de la page qu'on souhaite mettre en forme, on va structurer le contenu HTML à l'intérieur d'une balise `<div>` ayant une classe particulière.
On va définir en parallèle les règles de mise en forme CSS de ces différentes zones, l'identifiant faisant le lien. On pourrait aussi utiliser des classes.

5.1.4 Gabarit Twig

```
{% block body %}
<body>
   <div id="navigation">
   {% block navigation %}
   {% endblock %} {# navigation #}
   </div>
   <div id="menu">
   {% block menu %}
   {% endblock %} {# menu #}
   </div>
   <div id="main">
   {% block main %}
        <h1>{{ Message }}</h1>
   {% endblock %} {# main #}
   </div>
   <div id="footer">
   {% block footer %}
   {% endblock %} {# footer #}
   </div>
</body>
{% endblock %} {# body #}
```

Figure 8 – Structure de blocs Twig du gabarit de base

Dans les gabarits, on va définir des blocks surchargeables, correspondants au contenu des divisions, et portant les mêmes noms que les classes précédamment définies.
On a ainsi un nom de zone/block/classe unique pour faire le lien entre les différents éléments.

5.1.5 Standardisation / Spécialisation

Spécialisation dans une sous-section ou une page de certains éléments de structure ou de présentation

- héritage des gabarits : redéfinition du contenu de blocs
  ```
  {% block main %}
  ...
  {% endblock %}
  ```
- cascade des feuilles de style CSS : redéfinition de la mise en forme
  ```
  #main { background-color: lightblue; }
  ```

5.2 CSS avec Bootstrap

Bootstrap est un *framework* permettant de mettre en place un jeu de feuilles de styles CSS, en s'appuyant sur de mécanismes réactifs/adaptatifs.

5.2.1 Utilisation d'un *framework* de présentation CSS

- Standardisation du look
- Adaptatif
- Grille pour le positionnement graphique

— Intégration avec Twig / Symfony

5.2.2 Bootstrap

Framework CSS (+ JS)
`http://getbootstrap.com/`

5.2.3 Système de grille

Mise en page basée sur une grille de 12 colonnes

Figure 9 – Grille de mise en page dans Bootstrap

5.2.4 Exemples

— exemples, dans la documentation Bootstrap
— exemples sur *Start Bootstrap*

5.2.5 Thèmes

— Thèmes sur *Start Bootstrap*
— Feuille de syle additionnelle

Take away

— Structure != présentation
— Rôle de CSS
— Principe de surcharge des feuilles (cascade)
— Accessibilité
— Bootstrap

Annexes

Sur la qualité des interfaces Web, consulter le livre « *Qualité Web - La référence des professionnels* » - 2e édition - Eyrolles : `http://qualite-web-lelivre.com/` (disponible à la médiathèque)
Sur le sujet de l'accessibilité, vous pouvez également consulter la vidéo Voir autrement : l'ordinateur accessible

Postface

Crédits illustrations et vidéos

— Illustration Desktop Macintosh source Wikipedia
— Vidéo `https://www.youtube.com/watch?v=gc9NpYrbZgQ` par `UserExperiencesWorks`
— vidéo : Comment les aveugles utilisent-ils internet ? (Le nouvel obs)
— Image « HTML + CSS » empruntée au « CSS Tutorial » de Avinash Malhotra : `http://tutorial.techaltum.com/css.html`
— Diagrammes « *Specificity ranking by proximity* », « *Proximal specificity in action…* » et « *Specificity ranking by selector type* » empruntés à par Ernie Simpson, *aka The Big Erns* : ~~`http://cssway.thebigerns.com/special/master-item-styles/`~~ (site disparu)
— Diagramme de grille de mise en page de Bootstrap empruntée à Bootstrap de Twitter : un kit CSS et plus ! par Maurice Chavelli
— Illustration « Enjoy Websurfing », supposémment de Bruce Sterling

Copyright