Intégrations graphiques Publik - Autre #36831

CSS: définir une convention de nommage

10 octobre 2019 11:20 - Thomas Jund (congés, retour le 29/04)

Statut:	Nouveau	**Début:**	10 octobre 2019
Priorité:	Normal	**Echéance:**	
Assigné à:	Thomas Jund (congés, retour le 29/04)	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	Non

Description

Dans l'objectif de modulariser (isoler le code de chaque composant) et permettre leurs imbrications, j'ai envie de statuer sur une convention de nommage CSS.

Pourquoi ?

- Un composant ne doit pas dépendre de la zone du layout ou du bloc dans lequel il est inséré. un composant ".pk-button" doit pouvoir conserver son design qu'il soit injecté dans le header, le footer, la sidebar ou un autre composant (une cellule par exemple). Donc il faut en finir avec les selecteurs descendants du type #sidebar .mon-composant h2 {}
- CSS est global, oui c'est la merde mais c'est comme ça. Quand dans votre CSS vous définissez un style pour h2 {} hé ben tous les h2 de votre page seront impactés. Impossible de scoper une déclaration. Mais pour palier à ce défaut, augmenter le poids des sélecteurs est une très mauvaise idée pour la maintenance du code. Ça génère des cascades impossibles. Une fuite en avant. Sans parler performance.
- On a toutes les raisons de commencer sérieusement à faire du rangement, redécouper le code CSS en class unitaires, redécouper en plus petits fichiers. Gagner en lisibilité et faciliter la création de themes complexes. La transition vers les *web components* sera bien plus simple.
- Garder la possibilité, un jour, de basculer vers un framework JS de *web components*. Je ne dis pas qu'il faut, je propose de ne pas fermer la porte à la possibilité pour cause de migration trop complexe.
- Gagner en performance

Il y a plein de littérature sur le sujet et de conventions existantes: SMACSS, BEM (dans plein de variantes comme ABEM, BEMIT), Atomic design, …

Existants, contraintes et problèmes

- Il y des bribes de conventions existantes: .wcs/trucmuch/cell, .pk-button (prefix pk), .gru-content (prefix gru). Garder, jeter ?
- Ne pas casser l'existant, ce qui implique de conserver les id et class déjà en place, pour y ajouter la nouvelle convention, pas top; ou faire des rechercher/remplacer dans tous les thèmes existants :/
- Garantir qu'une fois la convention en place, la conserver et l'appliquer.
- Prendre en compte l'évolution de combo pour permettre aux CPF de personnaliser un thème depuis une UI.
- Garantir un convention suffisamment souple pour l'adapter aux évolutions et besoins futurs.

Quelques idées

- bannir les selecteurs par id dans le CSS core
- préfixer les blocs par applications
  - prefix 'pk-' pour tous les composants génériques type boutons, titres
  - prefix pour les blocs qui proviennent d'app spécifiques; prefix wcs-, auth-
  - prefix pour les blocs specific à un theme: .cd44-, .toodego-
- définir une class unique pour les éléments des blocs à la place d'un selecteur desc dans le core: .bloc-title {} à la place de .bloc .title {}

À trancher

- CamelCase ou trait-d-union ?
- Class unique, claire mais à rallonge comme .wcs-cell-currentDraft__title cell__title ou beurk plutôt crever.
- gestion des styles modificateurs ? .block--big ou block .is-big
- Gestion des styles par media-queries ? *s-col-1-1 m-col-1-2* ou *col-1-2@m*

Le débat est ouvert.

Demandes liées:		
Lié à Intégrations graphiques Publik - Development #24446: réduction de la sp...	**Nouveau**	**11 juin 2018**
Lié à Intégrations graphiques Publik - Development #13297: utiliser des noms ...	**Rejeté**	**24 septembre 2016**

Historique

#1 - 10 octobre 2019 12:17 - Frédéric Péters

Je trouve difficile d'isoler HTML et CSS, ce qui me semble être l'idée de se défaire des propriétés de la cascade; mais pour prendre la situation concrète, on a aujourd'hui une hiérarchie qui serait :

- le body, qui reprend des classes variant selon du paramétrage de l'admin, .page-template-standard .section-whatever
- des zones que l'admin peut remplir de cellules, ex: {% placeholder "content" name=name }, ces zones n'ont pas de règles concernant un élément englobant, on va avoir { placeholder "sidebar" } dans un , mais { placeholder "content" %} dans un , ou id="columns", ou autre, suivant les situations.
- des cellules/composants, ex: .textcell, .wcsformsofcategorycell, etc.
  - auxquels l'admin peut ajouter des personnalisations, ex: .dark-background, .highlight, .whatever
- dans ces cellules/composants, de l'html qui lui même peut partager certains éléments
  - ex: on trouve un class="links-list" pour les documents récents, pour les démarches en cours, pour les résultats de recherche, etc.
- et à quelques endroits l'admin peut écrire son propre HTML et on fournit quelques classes particulières pour lui donner un peu de souplesse, ex: pk-button.

Je fais peut-être un raccourci trop rapide, ou je rate un truc, mais c'est tout ça qui me dit qu'on ne peut pas se défaire de la cascade, parce qu'on continuera à vouloir qu'une cellule texte posée dans la zone "haut de page", adopte des couleurs différentes de celles d'une cellule texte posée dans le corps de page, parce qu'un titre dans cette cellule ne va pas avoir la classe template-standard--haut-de-page--textcell--title.

Bref la cascade, je ne vois pas comment faire sans, et comme la proposition par d'un "pourquoi ?" qui la met en question, je sèche.

~~

Anecdotique à côté oui il y a quelques éléments de conduite que l'on doit adopter pour simplifier les choses, réduire la spécificité des éléments communs (includes/*.scss), travail déjà entamé en réduisant la place des id, à continuer, des trucs élémentaires qu'il est en effet utile de noter, genre pas de camelCase.

#2 - 10 octobre 2019 17:31 - Thomas Jund (congés, retour le 29/04)

> Je fais peut-être un raccourci trop rapide, ou je rate un truc, mais c'est tout ça qui me dit qu'on ne peut pas se défaire de la cascade, parce qu'on continuera à vouloir qu'une cellule texte posée dans la zone "haut de page", adopte des couleurs différentes de celles d'une cellule texte posée dans le corps de page, parce qu'un titre dans cette cellule ne va pas avoir la classe template-standard--haut-de-page--textcell--title.

Pour moi il n'y a pas d'incompatibilité entre les besoins que tu exprimes avec l'application d'une convention de nommage qui limitera la cascade.
Je ne dis pas qu'il faut bannir la cascade. J'aimerais qu'elle soit utilisée uniquement si nécessaire et utile pour de l'héritage CSS, et pour éviter les conflits de styles dans les composants imbriqués et les modificateurs.
Tu exposes surtout les besoins d'appliquer des modificateurs sur les composants. Cela signifie-t-il que pour les styles propre aux composants on pourrait déjà simplifier les selecteurs avec une convention ?
Ensuite, pour les modificateurs, il est beaucoup plus souple pour un design system, qu'un composant soit modifié par une class "modificateur" qu'implicitement par le parent.

De plus, utiliser la cascade au niveau des styles custom d'un thème ne me pose aucun problème, le débat est pour les styles du core. Le core doit proposer un maximum d'options et imposer un minimum.

quelques exemples pour essayer d'illustrer mon propos

BAD

```
#sidebar .textcell {
  font-size: .8em
}
```

Dans cet exemple on impose un font-size avec un selecteur fort aux .textcell dans le core.
Et en cas de design différent pour le thème où on souhaiterais que de manière global toutes les .text-cell soient dans une font + grande un simple .text-cell { font-size: 1.2em } ne fonctionnera pas pour les textcell de la sidebar :(et il faudra en plus ajouter un selecteur specifique #sidebar .textcell@. Ce genre d'exceptions/bidouilles, appliquées à tous les composants d'un thème est vite un enfer et un code complexe inutilement.

GOOD

```
.is-small {
  font-size: 0.8em
}
```

Dans ce cas, il sera non seulement possible

- d'appliquer le modificateur où on veut (sur un composant, un élément d'un composant, ou directement sur la sidebar pour influencer tous les composants enfants et arriver au même résultat que précédemment),
- de ne pas imposer le style dans le core.
- et de donner la possibilité d'appliquer ce modificateur à travers l'UI de combo.

Il en va de même pour les éléments des blocs.

BAD

```
div.wcscurrentformscell ul > li > a {
  padding: 1rem;
}
```

Parce que si on décide (par exemple) de créer un nouveau modificateur .with-icon qui ajoute un icon en ::before ou ::after (avec entre autre les padding qui vont bien & qui est donc voué à pouvoir être utilisé sur n'importe quel lien ou Titre dans n'importe quelle cellule ou bloc).
Et bien ajouter cette class au selecteur précédent ne fonctionnera pas (toujours éventuellement à travers l'UI de combo). On est obligé aujourd'hui d'injecter ces styles à travers un extend sass dans chaque mega sélecteur spécifiquement et en surcouche ou alors de définir des variables $icon, $icon-position et écrire des conditions complexes dans les scss en multipliant les mega-selecteurs qui doivent accepter ces conditions.

GOOD

```
.wcs-cell-currentForms__a {
  padding: 1rem;
}
…
/* modificateurs */
.with-icon {
  padding: 2em
  &::before {content…}
}
```

L'objectif étant bien de faciliter la création de thèmes et d'apporter bien plus de souplesse aux possibilités graphiques tout en ayant un code plus simple à maintenir et à faire évoluer.
Je sais que c'est bien beau les généralités. Si tu veux on peut réfléchir au cas pas cas ce qui est applicable ou non et dans quelle direction, travailler sur des exemples précis.
L'objectif n'est pas de réécrire tout d'un coup. Mais pour pouvoir essayer, il faut quand même qu'on se fixe sur une première convention et migrer ce que l'on veut/peut progressivement.

#3 - 10 octobre 2019 22:50 - Frédéric Péters

> Pour moi il n'y a pas d'incompatibilité entre les besoins que tu exprimes avec l'application d'une convention de nommage qui limitera la cascade. Je ne dis pas qu'il faut bannir la cascade. J'aimerais qu'elle soit utilisée uniquement si nécessaire et utile pour de l'héritage CSS, et pour éviter les conflits de styles dans les composants imbriqués et les modificateurs.

Ok, je pense qu'on se retrouve là-dessus, c'est un point que j'avais mis de côté dans mon commentaire, en fait on devrait définir où sont les "frontières" des composants, et assurer que la cascade, appliquée, se fasse toujours par rapport à la "racine" du composant (et pas sur des connecteurs spécifiques à l'intérieur), avec toute la question de déterminer où commence/finit un composant.

Sur les id, on est d'accord sur l'élimination, sur le deuxième exemple, div.wcscurrentformscell ul > li > a serait en fait plutôt .links-list a, et dans la proposition .links-list--a, si je comprends bien, et du coup proposition de nommage qui s'applique surtout sur le dernier niveau de conteneur.

Le truc où je m'arrêtais, c'est que malgré tout ces sélecteurs "classe" qu'on utiliserait à différents niveaux de la hiérarchie, ils ont un certain poids, que les intégrations graphiques se trouveront devoir compenser, et cette compensation ne devrait pas se faire en "devinant", en se disant "ok j'ajoute un sélecteur classe je serai plus spécifique".

Bref, nommage ok, frontière aux composants, à déterminer.

#4 - 14 octobre 2019 15:34 - Thomas Jund (congés, retour le 29/04)

- Lié à Development #24446: réduction de la spécificité des CSS partagées ajouté

#5 - 16 octobre 2019 13:45 - Thomas Jund (congés, retour le 29/04)

> Le truc où je m'arrêtais, c'est que malgré tout ces sélecteurs "classe" qu'on utiliserait à différents niveaux de la hiérarchie, ils ont un certain poids, que les intégrations graphiques se trouveront devoir compenser, et cette compensation ne devrait pas se faire en "devinant", en se disant "ok j'ajoute un sélecteur classe je serai plus spécifique".

L'objectif est bien de maitriser la cascade en faisant en sorte que tous les composants et éléments aient tous le même poids. Soit celui d'une seule class (on va dire un poids de 10).

La cascade va donc se jouer majoritairement sur l'ordre du code.
Habituellement on ordonne les CSS du générique au spécifique et ça fonctionne bien. _custom arrivant en dernier pour les spécificités des thèmes, il ne peut que faire autant ou mieux qu'un poids de 10 (sauf pour les modificateurs forts) et donc prendre le dessus sans tracas.

On pourrait tenter de sez rapprocher d'ordre du type.

1. **reset** (on s'en fout on a pas).
2. **Styles globaux**, à travers les sélecteurs de balises directement (poids 1). Styles basics typo et formulaires le plus souvent. Permet de personnaliser la sous-couche d'un thème, fonts, tailles typographiques, puces de listes, couleur de sélection, styles par défaut des tableaux, etc. *_general.scss où on a pas grand chose* et *forms.scss* (même s'il y'a déjà du specifique)
3. **Utilitaires layout**: classes qui permettent de gérer le layout (grille) et des modificateurs de positionnement (sélecteur de class unique, poids 10). *grid.scss* et *utils.scss*
4. **Composants simples** : composants que l'on peut retrouver partout sur le site et dans d'autres composants plus complexes ou blocs: Liste des liens, boutons, tableaux, icons, spinner, class générique pour les cellules, etc. (sélecteur class poids 10). Les composants typographiques et de formulaires aussi. *library.scss*
5. **Composants complexes** : ici nos cellules individuellement. Idem; sélecteur de class unique. Si en dessous des composants simple, parce que possibilité de surcharger de modifier — soit par sélecteur de class simple ou sélecteur de class double (.cell__a.pk-button Poids 20) — le comportement d'un composant simple qui se trouve inclus. C'est à éviter au maximum, exceptionnel. Il est préférable de créer une variante de composant simple. (je peux donner un exemple si pas claire) ou de faire ça dans ***custom*** pour le theme en cours. _wcs.scss en virant la partie layout, *cells.scss*, carrousel.scss,
6. **Blocs** : Éléments uniques d'un template, souvent la/les navs, site header, site footer dans leurs différentes versions (peut comporter des composants simples). *nav.scss, _header.scss*, a11y.scss, user_info.scss
7. **Pages**: layout des pages (poids 10). *layout.scss*
8. **Modificateurs forts**. Ceux-là ils doivent s'appliquer quoi qu'il arrive. Du coup ils doivent faire peu de chose mais impérativement. Souvent on ajoute des !important comme ça on est tranquille, du coup on peut les ranger avec les utilitaires. Typiquement les class .hidden-*, .sr-only, .big, .small, etc. Ils sont rarement utilisés directement dans le code, mais injectés via JS ou via combo par exemple.
9. et enfin **custom** : surcharges spécifique à un theme. Dans la grosse majorité des cas, un simple sélecteur de class — vers un composant ou un élément d'un composant — permettra d'avoir le dessus. Une simple surcharge spécifique à l'aide d'une class du body .page-index .ma-cellule__element aura le dessus. Ici avoir des sélecteurs complexes avec des poids importants (#id > ol > li) ne posent techniquement pas de problème. Rien ne viendra surcharger cette surcharge. _

En faisant adapter progressivement l'existant on pourrait s'en rapprocher.
Dans l'idéal, quand l'élément d'un composant complexe ne surcharge pas un composant simple se trouvant intégré, l'ordre entre les points 4 et 7 ne sont pas importants.

#6 - 16 octobre 2019 15:01 - Thomas Jund (congés, retour le 29/04)

Concernant la convention en elle même.

Première proposition pour avancer.

Je propose BEM comme base, parce que la plus usuelle. à nous de l'adapter à nos besoins.

nom d'un bloc/composant/layout

1. bloc: Class des blocs
   - Le nom d'un bloc se compose du prefix de l'application qui le construit, suivi de son identifiant. L'identifiant doit être unique par prefix.
   - Le prefix est séparé de l'identifiant par un trait d'union (abusivement appelé tiret du 6): .prefix-composant
   - l'identifiant peut être composé de 2 parties séparé par 1 trait d'union. .prefix-composant-extra). L'utilisation du CamelCase n'est autorisé que dans le cas où l'identifiant du composant nécessiterait plus d'un trait d'union. .wcs-cell-currentDrafts
2. __element : Class des éléments
   - composé de la class des bloc, suivi de l'identifiant de l'élément, séparé par 2 underscores. .bloc__element .nav__list
   - l'identifiant des élements peut-être composé de traits d'unions. .bloc__sub-title
   - On ne peut pas coller 2 elements au nom d'un bloc pour définir un sous element: .nav__sub-nav__link n'est pas autorisé. Dans ce cas, définir un nouveau bloc sub-nav__link.
   - Dictionnaire d'éléments
     - __link
     - __item
     - __list
     - __title ou __tt
     - __body
     - __metas
     - __img
     - __wrapper
3. --mod : Options specifiques à un bloc
   - Si un modificateur (specifique à ce bloc, donc pas global) doit être appliqué à un bloc, son identifiant est ajouté après le bloc ou l'élément, séparé par 2 traits d'union. pk-button--rounded. Si la --modification engendre trop de CSS, il est préférable de créer un nouveau bloc. pk-rounded-btn

1. Class des modificateurs globaux. Ce sont des class qui peuvent être ajoutés sur n'importe quel bloc ou element pour le modifier, lui et/ou ses enfants (héritage oblige).

    - Pas de CamelCase
    - Pas d'underscore
    - trait d'union autorisé
    - prefix usuel encouragé .is-big, with-icon is-hidden is-visible, sr-only
    - --modificateur spécifique autorisé: with-icon--after

Exemple avec le composants steps sur lequel je planche en ce moment

- Blocs:
    - wcs-steps
        - Element
            - @wcs-steps__list
    - wcs-step
        - Elements
            - wcs-step__marker
            - wcs-step__marker-nb
            - wcs-step__marker-total
            - wcs-step__label

De cette façon, chaque élements peut être stylé à l'aide d'une seule class :

```
.wcs-steps {{ // styles du bloc}
.wcs-steps__list {{ // styles pour la liste (ol) }}
.wcs-step {{ // styles pour l'item (li) }}
.wcs-step__marker {{ // styles pour le marker (div) }}
.wcs-step__marker-nb {{ // styles pour le num du marker (span) }}
.wcs-step__marker-total {{ // styles pour le num total du marker (span) }}
.wcs-step__label {{ // styles pour le label (p) }}
```

Modifications d'état
3 états sont disponibles pour les step: current, before et after.
(Les états first et last sont obsolètes)
Idéalement il faudrait les appliquer comme --modificateur puisque l'état current est propre à step.

```
.wcs-step--current {}
.wcs-step--after {}
.wcs-step--before {}
```

Mais comme ça casserait les themes existant. J'ai utilisé.

```
.wcs-step.current {}
.wcs-step.after {}
.wcs-step.before {}
```

Et donc pour styler le marker d'un element .current, necessité d'un selecteur descendant

```
.wcs-step.current .wcs-step__marker {// }
```

Et là on monte déjà à poids de 30.

Convention et SASS

Avec SASS et une convention de ce type, il est possible d'utiliser le nesting pour éviter de réécrire le nom du block :

```
.wcs-step {
    &__marker {
        &-nb {}
        &-total {}
    }
    &__label {}
    // If step is current
    &.current {}
    &.current & {
        &__marker {}
        &__label {}
    }
    // If step is NOT current
    &:not(.current) & {
        &__marker {}
        &__label {}
    }
}
```

L'idée derrière une convention n'est pas qu'elle devienne un dogme. Il faut souvent accepter des exceptions ou des cas particuliers, surtout avec de

l'existant. Mais ça permet qu'on aille tous dans la même direction en ayant une bonne base et ça permet de gagner fortement en lisibilité du code.

Discussion ouverte aux amendements.

#7 - 16 octobre 2019 16:05 - Frédéric Péters

J'avais donc une inquiétude particulière sur les "frontières" de composant, l'idée qu'on veuille adapter le rendu d'une cellule selon qu'elle soit dans la barre latérale ou le pied de page, par exemple.

Mais ces cas sont finalement peu nombreux, j'ai juste noté ces trois-ci,

```
        div.textcell {
        /* don't include margins/borders for textcells embedded in
         * other cells (via extra placeholders) */
```

```
#footer .menucell {
    /* custom style for menu cells in footer, center links on a single line */
```

```
// style for introduction text cells on dashboard pages
@each $dashboard_class, $dashboard_image in $dashboard_items {
    .gru-content div.cell.info-text-#{$dashboard_class} {
```

qu'on peut laisser de côté, avec des résolutions particulières selon les cas.

#8 - 21 octobre 2019 15:15 - Serghei Mihai

Ça me va qu'on fasse du BEM, cela évitera trop d'imbrications dans les fichiers scss.

Pour isoler les composants par bloc, on pourrait peut-être imaginer l'attribution automatique des classes aux cellules.
Par exemple une cellule "Texte" dans le footer aura automatiquement la classe: footer-textcell. Et que toutes les classes définies au niveau de la cellule soient préfixées par le nom du composant: footer-textcell-adresse (pour une cellule texte du footer ou on mettrait l'adresse de la collectivité).

#9 - 23 octobre 2019 12:01 - Thomas Jund (congés, retour le 29/04)

fred n'étant pas fan des underscores, je propose que l'on fixe déjà que l'élément est identifié par 2 traits d'union à la place des 2 underscores.
Du coup, il faut donc trouver maintenant comment identifier les modificateurs.

Et je propose l'utilisation de **.
Ces caractères n'étant pas directement autorisés en nom de class par le parseur CSS, il faudra les échapper avec des antislashs.

```
<a class="pk-button**alert">
```

```
.pk-button\*\*alert {}
```

#10 - 23 octobre 2019 12:32 - Frédéric Péters

> fred n'étant pas fan des underscores

Pour référence mon problème c'est qu'ils m'apparaissent vraiment invisibles dans l'inspecteur firefox, quand une classe est sélectionnée, qu'ils se confondent donc trop facilement avec des espaces.

L'utilisation de ** et l'obligation d'échapper ça me semble dangereuse; ça ne tiendrait vraiment pas d'également utiliser deux tirets ?, je ne vois vraiment pas les situations de conflits qui empêcheraient ça.

#11 - 23 octobre 2019 14:52 - Thomas Jund (congés, retour le 29/04)

Le risque étant de ne pas clairement identifier ce qui est élément de ce qui est modificateur (il faudrait parler de variation ou variante).

Certains dev front n'utilisent pas les variations dans la même class que l'élément, ils préfèrent utiliser une nouvelle class prefixé avec un trait d'union.
J'ai n'ai jamais essayé cette convention mais pourquoi pas, on verra au fur et à mesure les impacts et on ajustera.

```
// À la place de
.block--modificator
// Utiliser
.block -modificator
```

Utiliser un trait d'union en prefixe est standardisé depuis l'arrivé des prefix navigateurs.

en CSS on aurait donc

```
-small {}
// ou
```

```
.block--element.-small {}
```

Références
https://webuild.envato.com/blog/chainable-bem-modifiers/
https://css-tricks.com/abem-useful-adaptation-bem/

#12 - 23 octobre 2019 15:28 - Frédéric Péters

> Le risque étant de ne pas clairement identifier ce qui est élément de ce qui est modificateur (il faudrait parler de variation ou variante).

Oui mais dans la pratique, je n'ai pas l'impression que ça puisse arriver; mais peut-être à nouveau je visualise mal ce qu'on a qui pourrait être "modificateur".

#13 - 23 octobre 2019 16:38 - Thomas Jund (congés, retour le 29/04)

Difficile de te donner un exemple qui colle bien au projet.

Mais bon. J'essaie.

On a une navigation qui existe en plusieurs variantes en mobile.

- une version hamburger
- une version bottom-bar

Plutôt que de gérer ces différences avec des vars sass, on décide d'appeler un template django différent à travers une UI dans le BO de combo.

On aura donc 2 variantes.

```
<nav class="nav--hamburger" />
  ou
<nav class="nav--bottom-bar" />
```

Mais comme la nav burger a un element bouton `hamburger`, il peut y avoir confusion à la lecture du code entre selecteur pour l'element hamburger ou variante hamburger.
Autre exemple déjà croisé: HTML a un tag <small> et j'ai déjà croisé un selecteur BEM `block__small` pour selectionner l'élément small du block. Le dev a fait un choix sémantique très fainéant, et discutable, OK, mais au moins je savais que small était un element. Avoir une sémantique fine n'est pas donnée à tous.
Et dans la vie d'un projet qui passe par plusieurs devs, et encore plus dans un projet libre ouvert à contributions, avoir un system safe est important non ?

Moi je trouve ça bien mieux d'avoir une différence typographique pour identifier le rôle d'une class. Après, si on trouve une solution avec des tirets uniquement. Pas de problème.

#14 - 23 octobre 2019 17:00 - Frédéric Péters

> Et dans la vie d'un projet qui passe par plusieurs devs, et encore plus dans un projet libre ouvert à contributions, avoir un system safe est important non ?

Pour moi bien moins important que la lisibilité; et je préfère un système exceptionnellement ambigu mais généralement lisible à un système jamais ambigu mais généralement illisible. (et comme je me bats déjà contre mon impression que tout ça crée un truc illisible et contre nature, toutes ces propositions me remettent ça devant les yeux)

#15 - 23 octobre 2019 17:38 - Frédéric Péters

J'essaie de continuer à réfléchir et pour ça je vais tenter de résumer, en tapant ... comme séparateur, pour illustration.

On a la structure générale bloc...element...option.

On peut avoir des styles bloc...element, exemple navigation...item.

On peut avoir des styles bloc, exemple navigation.

Il pourrait également y avoir des styles bloc...option, exemple navigation...horizontal.

Le soucis est qu'on peut alors confondre bloc et élément, item et horizontal.

Deux vagues idées,

- imposer que pour spécifier une option il faut nécessairement bloc et élément; se lève alors la question d'appliquer une option à la racine du bloc; pourquoi pas alors avoir un "pseudo-élément" root, exemple navigation...root...horizontal. Ainsi on sait que deux morceaux ce sera toujours bloc & élément, et que trois morceaux ce sera toujours bloc & élément & option.

- imposer un préfixe aux options, ex: navigation...option-horizontal.

#16 - 23 octobre 2019 18:02 - Thomas Jund (congés, retour le 29/04)

En fait ce que BEM appel modificateurs est un truc fourre tout où on peut trouver.

- information d'état (state) : `current`, `active`, `hidden`, …
- information de position : `first, `last`, …
- modificateurs graphiques : `red`, `big, `small`, …
- des variantes: `horizontale`, `vertical`, `on_top`, etc.

Actuellement, sur les Steps par exemple, les états (current, step-before, step-after) et les positions (first, last) sont gérés par des class supplémentaires.

Je propose de statuer pour le moment sur le modèle

block--element

Et qu'on prenne une décision sur les états (class supplémentaire, préfix specifique, etc) que je puisse boucler les steps et de décider pour le reste quand le besoin arrivera. Sur des cas concrets on arrivera peut-être mieux à trouver un terrain d'entente.

Que penses-tu de la proposition précédente avec une class séparée + tiret en prefixe pour les états ?

#17 - 23 octobre 2019 18:10 - Frédéric Péters

Ok pour avancer sans ça pour le moment et voir quand ça deviendra plus concret. Sur les classes commençant par un tiret, outre le côté bizarre, ça me donne l'impression de casser l'effort mis à tout taper dans une unique classe pour garantir un seul niveau de spécificité.

#18 - 13 novembre 2019 10:16 - Thomas Jund (congés, retour le 29/04)

Bon, en bossant sur les steps, je n'ai pas trouvé de solution pour les class d'états.
Je propose de les laisser séparer de la class block--élement, ce qui reste logique dans une optique d'ajout et suppression à l'aide de JS .classList().
Après, est-ce qu'on les préfixe avec le nom du bloc si ce sont des états spécifiques à l'élément en cours.

Par exemple pour les steps on aurait donc 3 class d'états

```
.wcs-step-before
.wcs-step-after
.wcs-step-current
```

avec le prefix block `wcs-step-` pour ne pas entrer en conflit avec une possible class `.current` par exemple.

en Html on aurait donc

```
<li class="wcs-step wcs-step-current">
```

ce qui nous permet d'utiliser directement en css le selecteur

```
.wcs-step { // design par défaut de la step }
.wcs-step-current { // ajout des styles spécifiques à la step current }
```

ou alors ne pas préfixer les class d'état, même lorsqu'elles sont spécifiques à un composant.

```
.after
.before
.current
```

en HTML

```
<li class="wcs-step current">
```

Et dans ce cas devoir utiliser des selecteurs multiples pour styler les états.

```
.wcs-step { // design par défaut de la step }
.wcs-step.current { // ajout des styles spécifiques à la step current }
```

Possible aussi, mais pour éviter le risque de conflit avec une class .current définie en amont il faudrait alors décider d'un préfixe pour les class d'état globales et s'y tenir.

La première proposition est quand plus verbeuse mais plus safe.

#19 - 29 novembre 2019 16:46 - Thomas Jund (congés, retour le 29/04)

Premier draft disponible
https://dev.entrouvert.org/projects/prod-eo/wiki/HowDoWeDoCSS

N'hésitez pas à faire des retours.

#20 - 04 décembre 2019 14:17 - Thomas Jund (congés, retour le 29/04)

- Lié à Development #13297: utiliser des noms de classe/d'id spécifiques publik ajouté

#21 - 27 janvier 2020 18:24 - Frédéric Péters

Rencontré dans Development #39091: nouvelle cellule "Pages récemment modifiées", lastpagesupdatescell--etc.; ça m'irait bien d'ajouter dans les conventions que les mots restent séparés, c'est sinon vraiment trop compliqué, je trouve, à lire.

#22 - 28 janvier 2020 09:27 - Thomas Jund (congés, retour le 29/04)

+1.
Je l'évoquais justement avec nico hier: cquoicesclassillisibles.
Et je vois que tu as eu la même réflexion en créant #39314.

#23 - 29 janvier 2020 10:58 - Thomas Jund (congés, retour le 29/04)

Ajout d'une recommandation sur l'utilisation des traits d'union dans l'identifiant CSS d'un block pour raison de lisibilité
https://dev.entrouvert.org/projects/prod-eo/wiki/HowDoWeDoCSS