HowDoWeDoThemes

(work in progress)

Le dévéloppement d'une intégration graphique exige un environnement de développement local, cf https://doc-publik.entrouvert.com/dev/installation-developpeur/

publik-base-theme

Le module publik-base-theme malgré son nom est le dépôt de (presque) toutes nos intégrations graphiques.

La liste est établie (lors de l'appel à make) en parcourant les répertoires sous static/, en tirant les informations de fichiers config.json, ex static/publik/config.json :

```
{
  "label": "Publik",
  "variables": {
    "theme_color": "#E80E89"
  }
}
```

- Un identifiant est créé automatiquement en prenant le nom du répertoire.
- Le libellé sert à la sélection du thème dans l'écran dédié dans Hobo.
- La variable theme_color reprend la couleur dominante du thème, elle est affichée dans l'écran dans Hobo mais également posée dans des balises <meta> pour colorer l'UI (chrome) de certains navigateurs web (mobile).
- Il peut aussi y avoir une variable favicon, contenant le chemin vers la favicon (ex: fondettes/favicon.png pour /static/fondettes/favicon.png).
- Une variable logo_link_url pour que le lien primaire du titre/logo de la page ne pointe pas vers le portail citoyen mais vers ailleurs.

Style CSS

Dans publik-base-theme/static, il y a donc un répertoire par intégration. Celui-ci doit contenir un fichier style.css. On utilise le préprocesseur sass pour le créer à partir d'un fichier style.scss, ce travail est assuré via des règles dans le Makefile, la cible pour la création des fichiers style.css s'appelle "css" → make css exécuté depuis le répertoire publik-base-theme créera les fichiers style.css des différentes intégrations. Il est bien sûr également possible de viser un seul fichier, ex: make static/publik/style.css.

Le fichier style.scss sert à importer différents fichiers, généralement :

```
@import 'vars';
@import '../includes/publik';
@import 'custom';
```

La définition des styles de Publik vient de ../includes/publik, les deux autres fichiers servent à définir les spécificités locales.

Le fichier _vars.scss peut définir une série de variables (liste complète publiée sur https://doc.entrouvert.org/publik-base-theme/dev/misc-scss.html) qui permettent de paramétrer le rendu général du site, généralement en y appliquant les valeurs tirées d'une maquette ou du site servant de modèle.

```
@charset "UTF-8";

$primary-color: #1DA1AE;

$font-color: #565656;
$font-size: 13px;
$font-family: sans-serif;
$nav-background: $primary-color;
$nav-color: white;
$nav-active-color: darken($primary-color, 20%);
$border-radius: 3px;
$button-background: $primary-color;
$title-background: $primary-color;
```

```
$title-color: white;
$footer-background: transparent;
$footer-color: $font-color;
```

Pour aller au-delà de ce que ce paramétrage permet; le fichier _custom.scss peut accueillir des règles CSS supplémentaires :

```
#header {
    background: url(images/bandeau.jpg) no-repeat scroll left bottom;
    height: 180px;
}

body {
    background: url(images/bg_header.png) repeat-x scroll 0 2px;
}
[...]
```

Si on se trouve à copier/coller ou répéter des trucs de _custom.scss en _custom.scss, on doit sans doute réfléchir et voir pour intégrer ça dans les fichiers scss partagés.

Documentation publique des classes css

- à lire ici au préalable pour ne pas doublonner de l'existant : https://doc-publik.entrouvert.com/admin-fonctionnel/modifier-le-contenu-des-portails/

Compilation dynamique et livereload

En développement, il peut être pratique de relancer la compilation des thèmes lorsque des fichiers sont modifiés et de rafrchair les pages correspondantes dans le navigateur automatiquement. Pour cela, vous pouvez utiliser le script ./compile-and-reload.sh. Pour que cela fonctionne, assurez vous que votre environnement de développement est à jour (https://doc-publik.entrouvert.com/dev/installation-developpeur/#Mettre-%C3%A0-jour-son-installation) et que ~/.config/publik/settings/combo/settings.d/local.py contiennent bien LIVERELOAD_ENABLED = True.

D'autres n'utilisant pas livereload sont présentées ci-dessous :

```
# La première fois, penser à lancer aussi:
#   sudo apt install iwatch
iwatch -r -t '.scss' -e modify -c "make css" .
```

Lorsque cette commande est active, chaque modification d'un fichier SCSS dans le projet déclenche une recompilation du ou des thèmes correspondants.

Alternativement le script https://git.entrouvert.org/misc-fred.git/tree/bin/sassw assure ce taf.

Pour rafraîchir automatiquement le CSS dans le navigateur, vous pouvez installer cette extension Firefox : https://addons.mozilla.org/en-US/firefox/addon/content-override-watch/

Pour la configurer, installez l'application hôte en python qui charge le contenu local :

```
pip install cow-web-ext
cow-web-ext install
```

Puis configurez les mappings vers les fichiers dans les paramètres de l'extension dans firefox (Settings => Extensions & Thèmes => Content Override & Watch). Pour par exemple charger le CSS depuis votre dossier publik-base-theme :

```
.*\/static\/([^\/]*)\/style.css  => ~/src/publik-base-theme/static/$1/style.css
```

Utilisez l'icone dans la barre d'outils pour activer / désactiver la synchronisation avec le CSS local. Ça fonctionne aussi sur les sites en recette & prod, et permet de tester les changements de CSS directement sur du contenu de production.

Templates

Les templates Django sont dans le répertoire templates/; ils surchargent une partie des templates natifs des différents modules, pour diverses raisons (spécificités à Publik, paresse, etc.).

À éviter, il y a la possibilité pour un thème, de redéfinir de manière spécifique un template donné, en le plaçant sous templates/variants/$theme/, par exemple templates/variants/nancy-2017/combo/wcs/forms_of_category.html.

Images

Elles sont normalement rangées dans le répertoire img/ (mais ce n'est pas encore systématique).

Pour des petites images, il peut être intéressant d'optimiser le chargement en incluant directement l'image dans le fichier CSS, en utilisant une URI data. Pour assurer ce travail il y a une cible data_uris dans le Makefile, qui appelle le script make_data_uris.py sur un répertoire donné, ex :

```
python make_data_uris.py static/grandlyon-gnm/
```

Cela crée un fichier _data_uris.scss qui peut être inclus dans le thème et les images peuvent alors être référencées ainsi : url($data_uri_cadre_de_vie) (data_uri_ + le nom de fichier sans l'extension).

Icônes du tableau de bord

Inclure ../includes/dashboard dans les scss et ajouter le nécessaire au Makefile pour générer les images dans les couleurs désirées, ex :

```
cd src/ && python render-imgs-dashboard.py ../static/chateauroux/img/ --normal 333333 --selected 0
779B7 --title FFFFFF --title-width 80
```

Icônes des catégories

Inclure ../includes/categories dans les scss et ajouter au Makefile.

```
cd src/ && python render-imgs-categories.py ../static/orleans/img/ --primary f05923 --secondary 34
697D
```

Polices de caractère

On évite la récupération de polices depuis les CDN de Google ou autre, on incorpore directement dans publik-base-theme les polices (pour lesquelles les licences autorisent cela, bien sûr). On se trouve ainsi dans static/includes/ avec une série de fichiers _font-*.scss, ex: _font-montserrat.scss.

Pour donner accès à une police, on ajoutera donc un import de la forme :

```
@import '../includes/font-montserrat';
```

On inclut uniquement les versions woff2 et woff car c'est désormais pris en charge "partout" (>IE8).

Détails de la mécanique bas niveau

(Cette section n'est pas nécessaire pour la réalisation de thèmes.)

On définit un thème pour Combo ; d'abord de manière classique, par exemple un "common.html" qui va de <html> à </html> et puis les différents modèles de page, "page_template.html" et "page_template_sidebar.html" etc. qui {% extends "common.html" %}.

On précise les URL vers des ressources (css, js, images…) de manière absolue, il y a pour ça un {{site_base}} qui est utile; ex :

```
<script src="{{site_base}}{% xstatic 'jquery' 'jquery.min.js' %}"></script>
<script src="{{site_base}}{% static 'js/combo.public.js' %}"></script>
```

Ensuite, les fichiers base.html des applications peuvent être ajoutés (idéalement les applications cherchent toutes d'abord un /$app/base.html, donc les templates ne se mangent pas l'espace du /base.html). Ces fichiers démarrent sur un modèle simple, d'une ligne, {% extends theme_base %}. Cela créera à la volée un template avec un squelette généré sur base de celui servi par combo.

Pour ce faire l'application (en fait un context processor d'hobo) se base sur un paramètre THEME_SKELETON_URL, qui pointe vers le combo, ex :

```
THEME_SKELETON_URL = 'http://combo-test-theme.127.0.0.1.xip.io:8000/__skeleton__/'
```

Le context processor ajoute l'url demandée (paramètre source) et avec ça Combo trouve la page correspondant à l'application (en matchant le mieux possible l'url donnée, ce qui permet par exemple d'avoir une page pointant vers http://wcs.example.net/etat-civil/ et une autre pointant vers http://wcs/sport/, et y avoir des différences de contenu). Combo fait le rendu de cette page en remplaçant les zones laissées vides (sans cellules) par des appels de bloc django (ex : {% block content }{ endblock %}), ceux-ci seront alors

interprété quand le context processor créera un objet Template() à partir du contenu de la page ainsi récupérée.

C'est possible d'ajouter des zones qui donneront des blocs sans pour autant donner des zones présentées à la composition des pages dans combo, il faut utiliser le templatetag skeleton_extra_placeholder, ex : {% skeleton_extra_placeholder 'extra-head' } ou { skeleton_extra_placeholder 'extra-body-args' %}; c'est ainsi possible de créer une structure correspondant à celle attendue par l'application.

Elle pourrait être composée directement au niveau de combo mais ça peut alourdir mochement le code, par exemple si on a authentic qui attend body-args et une autre qui atend autre-body-args, et ainsi de suite :

```
<body {% block body-args %}{% endblock %} {% block autre-body-args %}{% endblock %} ...>
```

Pour éviter ça, on peut mettre un seul bloc, et dans le $app/base.html, prolonger le extends par un mapping des blocs :

```
{% extends theme_base %}
{% block extra-body-args %}
  {% block bodyargs %}
  {% endblock %}
{% endblock %}
```

Définition intégration "light"

- pas de template
- un _custom.scss de max 100 lignes

Mise en recettes et release du paquet deb publik-base-theme

Ça se passe dans de dépot publik-base-theme

1. un fois ses commits validés, on peut pousser sur la branche principale (main)
2. Jenkins va faire construire à partir de cette branche, à intervalles réguliers, et créer un paquet deb dans le dépôt 'unstable' d'EO
   - vérifiez que tout se finit bien ici : https://jenkins.entrouvert.org/job/publik-base-theme-deb/
3. si les autres développeurs sont d'accords, on peut taguer une nouvelle version du dépôt Git : git tag -a v2.22
4. Jenkins va aussi construire à partir du tag et créer un paquet de dans le dépôt 'testing' d'EO
   - vérifiez aussi que tout se passe bien
5. on peut alors mettre à jour les recettes sur **toutes les machines**, on peut exécuter apt-get update && apt-get upgrade publik-base-theme
   - liste de toutes les machines de recette
   https://dev.entrouvert.org/projects/sysadmin/wiki/Mises_%C3%A0_jour_Publik#Recettes
   - on peut utiliser l'outil eotasks pour exécuter les commandes en parallèle sur les serveurs:
   [[sysadmin:Exécution_de_taches_sur_les_serveurs]]