Enllaç al lloc web amb la pràctica:

https://html-css.byskat.me/pra/

Sota aquest subdomini es troben les demés PACs desplegades utilitzant Netlify.

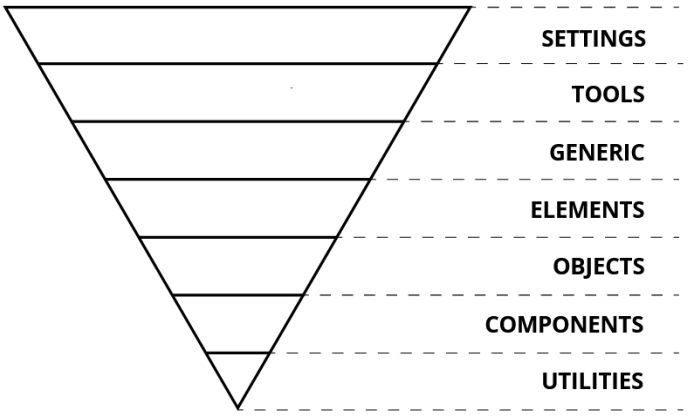
Documentació

L'únic arxiu CSS es troba dividit en tres blocs o seccions que defineixen en diferent especificitat els estils de la pàgina:

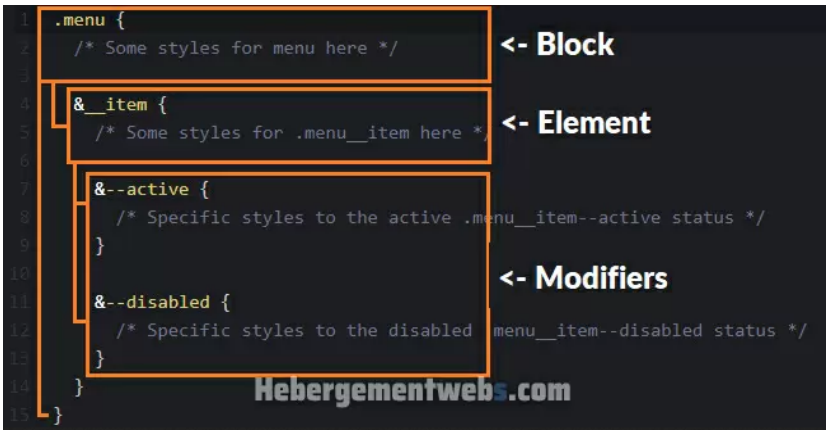
- Elements per defecte (tags html sense clase)
- Clases utilitàries (tot i que s'estenen lleujerament del significat)
- Clases de component

Per tant, una de les millors maneres de refactoritzar el codi css en cas d'extendre el projecte, passaria primer de tot en dividir les seccions en fitxers únics i col·locant-los en ordre en un fitxer a mòde d'índex utilitzant la regla @import. Això permetrà no només organitzar més eficientment el codi, si no també, evitar importar codi innecessari (els components que a priori apareixen a l'apartat de preus no té perquè carregar a l'inici).

La imatge que veiem a la dreta es el model del triangle invertit o *Inverted Triangle CSS* (ITCSS) que ordena les regles css així com les classes de tal manera que a la posició més elevada (o els primers en ser importats) son els elements més genèrics fins als més específics, això evita conflictes durant l'ampliació d'un projecte tot utilitzant la pròpia naturalesa en cascada de CSS.



Aquesta aclaració és important no només per entendre com ampliar el projecte, si no per entendre com es divideix actualment. Tal i com apareix enumerat anteriorment hi ha classes utilitàries que encompassen des de les variables (:root), els elements (declaracions per defecte dels tags), objectes com la classe `.wrapper` i components com `.pricingCards`. Aquestes classes es troben degudament dividides (amb comentaris separadors). És convenient dir, que l'actual estructura tot i intentar seguir l'equema definit, no deixa de ser una versió simplificada adaptada al projecte.



L'altre element "arquitectural" a tenir en compte es la metodología de denominació de les pròpies classes, que com en altres activitats torna a ser la que proposa BEM. Es a dir, que cada component (o objecte) compta amb una clara jerarquía i ús concret.

Aquesta metodologia no deixa de ser una convenció que no té perquè ser millor que cap altre, però precisament al aplicar un sistema existent i funcional (i **documentat**) a l'hora d'anomenar les classes entenem les diferències intuitivament i la relació entre elles, per exemple: `.features` (un component que apareix a l'índex) amb `.features__header`.

Ara bé, això també dona espai a la millora, ja que tot i escriure les classes amb aquesta sintaxis resulta positiu per si mateix, fer-ho amb preprocessadors tals com SCSS resulta encara més convenient al permetre l'aniuament de les classes i l'ús de **&** o selector del "pare" (tal i com es veu a la imatge anterior).

Objectes

Un dels principals objectes del projecte es el `.wrapper` aquest envolta tots aquells elements que segueixin l'estructura de la pàgina ja que marca el màxim que aquest es pot desplegar o l'espai de separació entre el contenidor i el límit de la pantalla.

Un altre objecte rellevant es el selector `main.page` ja que es força a col·locarlo sempre quan comença el contingut principal, que també es on es determina on comença el contingut únic o principal de la pàgina `id="mainContent"`.

Components

Hi ha una llarga llista de components, però en destacaré dos que son els que apareixen a totes les pàgines:

`.navMain` aquest element conté el menu "sticky" amb el logo, es únic i es el primer que ens trobem al obrir el <body>.

`.footerMain` de la mateixa manera que l'anterior, aquest component no es repeteix més que un cop per pàgina i fa referència al final de la pàgina que conté

Ambdós components reben el .wrapper a l'interior (també el .hero) fent que el fons (determinat per la classe .accentedStyle) ocupi la pantalla d'extrem a extrem.

Es destacable dir, que els components s'entenen com a elements autocontinguts i que les seves classes no afecten a elements externs a ells. Per evitar repeticions hem d'abstreure les similituds o "coincidències" en altres classes o components.

Altres classes importants

`.accentedStyle` s'utilitza exclusivament per modificar els colors d'una secció, actua com una classe temàtica. Es pot observar, per exemple, amb el menú principal o el .hero.

`.flatList` classe utilitaria que literalment "aplana" les llistes <ul> i els seus fills per a aplicar-li -posteriorment- nous estils.

`.ico` la solució per a afegir icones al footer: en comptes de crear un pseudoelement amb id's o variants amb contingut específic, s'ha utilitzat un atribut que conté el códi (normalitzat) del símbol i amb la funció attr() se'n fa referència dins el css. Fent aquest canvi podem reutilitzar la classe i generar diferents tipus d'icones sense modificar el CSS. Per alguna raó (que honestament, no acabo d'entendre) la ícona de Facebook dona warning al validador, tot i que sembla no ser únic ni tampoc greu.

`.btn` tècnicament un component tot i que posicionat en la primera secció (no n'és l'ùnic) fa referència a tots aquells botons o enllaços que apareixen arrodonits, en aquest cas només té una variant --large, tot i que hauria d'haver-hi una variant transparent (que apareix a la pàgina de preus), i que per ~~desídia~~ falta de temps s'ha deixat així.

Estructura bàsica

En cas de crear una nova pàgina s'ha de seguir l'estructura indicada a l'imatge, tot modificant la classe `.active` del menú (indicant correctament a quina pàgina ens trobem).

Dins del tag `<main>` hi col·loquem el contingut de la pàgina, incloent en algún punt (si és necessari) la classe `.wrapper` descrita anteriorment.

```
<!DOCTYPE html>
<html lang="en">

<head>···
</head>

<body>
  <nav class="navMain accentedStyle">···
  </nav>

  <main id="mainContent" class="page">···
  </main>

  <footer class="footerMain accentedStyle">···
  </footer>
</body>

</html>
```