PAC3

M4.258 - EINES HTML I CSS II

GitHub: https://github.com/stratdi/M4.258-PAC3
Netlify: https://sparkling-pie-41cae6.netlify.app

Jordi Cruz Lladó

Taula de contingut

Procés de desenvolupament de la pràctica

Creació i configuració inicial de repositori

Per crear el repositori hem anat a la nostra pàgina de GitHub, i dins la secció Repositories i hem clicat damunt 'new'.

Com a nom del repositori, hem posat el codi de l'assignatura i la PAC pertintent (M4.258-PAC2).

Seguidament, hem anat al repositori de la UOC Boilerplate i hem davallat el codi amb un .zip.

Un cop realitzades aquestes passes, hem clonat el nostre repositori al PC local amb la següent comanda:

```
git clone git@github.com:stratdi/M4.258-PAC3.git
```

Com que ja tenim la clau pública/privada configurada, no ha fet falta realitzar les passes de nou.

Un cop clonat el nostre repositori, hem volcat el contingut del .zip davallat amb la UOC Boilerplate, i hem executat les comandes típiques per enviar els canvis al servidor.

```
git add *
git commit -m "Boilerplate importat"
git push
```

D'aquesta manera ja tenim el punt inicial de la pràctica.

Entorn de desenvolupament

Hem utilitzat el mateix entorn de desenvolupament que per altres PAC, Visual Studio Code. Com que ja es troba instal·lat, només l'hem hagut d'obrir.

Remarcar que, al fer feina professionalment amb l'IDE Eclipse, tenim instal·lat el plugin d'Eclipse keymaps, per tenir les dreceres de teclat típiques d'Eclipse al VSCode.

Comprovam que tenim NodeJS instal·lat (per fer servir l'npm) amb la següent comanda:

```
npm -v
```

Ens apareix que tenim instalada la versió 10.1.0, per tant no cal instalar de nou NodeJs.

Dependències de la UOC Boilerplate i afegides

Ara és moment de revisar el fitxer package.json per veure les dependències que té la UOC Boilerplate i, en cas de no voler alguna, eliminar-la. També, és moment d'afegir els scripts que vam utilitzar a la PAC1/PAC2 per automatitzar tasques en la compilació.

Les dependències de desenvolupament (devDependencies) que inclou les necessitam, ara bé, n'instal·larem de noves per poder desenvolupar la plana web amb Tailwind.
Per això, instal·larem Tailwind. Ho farem executant les següents comandes:

```
# Instal·lació de depencència
npm install tailwindcss --save-dev

# Per generar el fitxer inicial de Tailwind.
npx tailwindcss init

# Per instal·lar PostCSS per emprar-lo amb Parcel
npm install postcss --save-dev
```

Cream el fitxer anomenat .postcssrc per definir el plugin de Tailwind. Hi posarem el següent contingut:

```
{
  plugins: {
    tailwindcss: { }
  }
}
```

Ara, obrim el fitxer de configuració de Tailwind i al node 'content' li afegim el següent, de manera que sàpiga on aplicar Tailwind:

```
module.exports = {
  content: ["./src/**/*.{html,js}"],
  plugins: [],
}
```

Un cop instal·lat Tailwind, obrim el fitxer principal de CSS (main.scss) i li afegim el següent:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

D'aquesta manera ja podrem emprar Tailwind correctament.

Ara ha arribat el moment d'instal·lar dependències/plugins que volem emprar al lloc web. Per aquesta PAC, emprarem FontAwesome Free i PostHTML-expressions. Amb PostHTML-expressions podrem emprar directives de programació, com loops o switch, i amb FontAwesome Free podrem visualitzar icones de manera ràpida i intuitiva.
Els instal·lam amb les següents comandes:

```
npm install posthtml-expressions --save-dev
npm install @fortawesome/fontawesome-free --save
```

Observam com al package.json s'han afegit les noves dependències a un nou node del JSON.
En apartats següents explicarem com hem fet ús d'aquestes dependències.

PostHTML-expressions

Com hem indicat a l'apartat anterior, aquest lloc web farà ús d'aquest plugin.
L'hem emprat per poder crear "components" emprant PostHTML. Concretament, per crear el component "list".

Gràcies al plugin, podem fer un template que reb un array per paràmetre i iterar-lo, de manera que podem reutilitzar aquest codi i personalitzar-lo.

FontAwesome

Utilitzarem aquesta dependència per posar les icones de xarxes socials al footer.

Desenvolupament

Una vegada tot configurat, executam la comanda per instal·lar tot el necessari pel lloc web i executar-la:

```
npm install
npm run dev
```

Publicació del lloc web

La publicació a Internet es pot dir que és la passa final del desenvolupament web. Això implica la posada en línia d'un lloc web o contingut digital perquè sigui accessible per a una àmplia audiència.

Hi ha diferents formes de publicació d'una pàgina web, però nosaltres utilitzarem l'eina especificada a l'enunciat de la PAC: Netlify. Netlify ens permet, a partir d'un repositori de codi Git, poder automatitzar la publicació de la pàgina web a la xarxa. Té diferents avantatges que hem de tenir en consideració: desplegament continu de les modificacions de la pàgina web, certificat SSL gratuït, i el pla gratuït al qual ens acollim.

D'aquesta manera, hem pogut publicar la pàgina web sense cap complicació i, gràcies a l'anterior, fer correccions i veure com es desplega quasi al moment.

La configuració ha estat molt senzilla: hem fet login amb el nostre compte de GitHub i hem donat permís per accedir als repositoris. Després, hem seleccionat el de la PAC3 i com a comanda d'execució hem posat la següent:

```
npm run build
```

Així executarà l'script del package.json 'build'. A partir d'ara, per cada commit que facem, Netlify compilarà i desplegarà el nostre lloc web de manera automàtica.

Decissions estratègiques

Ús de templates amb PostHTML

Seguint amb la filosofia DRY, en vers de repetir codi comú a tots els fitxers HTML, utilitzarem templates per tenir el codi autocontingut i inclou-re'l a cada pàgina que ho necessiti.

En aquesta PAC, sense contar footer i menú, hem creat dos templates:

- **Card**: típica carta que conté una fotografia a la presentació i davall un títol i un text. Aquest template espera per paràmetre la següent informació:
  - **Imatge**
  - **Nom**
  - **Descripció**
- **List**: llista d'elements ordenats. Consta d'un índex numèric, un títol i una descripció. Aquest template espera per paràmetre un array d'objectes amb la següent informació:
  - **Títol**
  - **Descripció**

El template de Card l'emprarem per representar els jugadors del Top 6, i el template de List l'emprarem per llistar activitats a la pàgina de Qui som.

Reutilització d'estils

Tailwind ens permet reutilitzar estils quan, per múltiples nodes, empram sempre les mateixes classes de Tailwind per donar estil a un component.

En aquesta PAC, hem emprat aquesta reutilització de la següent manera:

- **Classe 'bg-radial'**: aquesta classe empra directives de Tailwind per generar el fons de pantalla radial de color verd, ja que s'utilitza a cada pàgina i així és més senzill definir-ho.
- **Classe 'card'**: aquesta classe representa un element 'card', però no de jugador, sinó el típic contenidor rodó amb ombra i de color blanc. A la pàgina de 'Qui som' hi ha molts de components que empraven les mateixes i, d'aquesta manera, es simplifica l'ús.

Imatges i propietat intelectual

Totes les imatges del lloc web han estat creades amb l'intel·ligència artificial Bing (Bing Image Creator).

Preguntes utility-first CSS

Quines diferències hi ha entre l'enfocament de tipus CSS semàntic i el CSS d'utilitats? Com ha afectat això el teu procés de desenvolupament? I el teu codi?

El CSS semàntic es centra en l'ús de noms de classes i identificadors que reflecteixen el significat i la funció dels elements HTML, prioritzant la llegibilitat i mantenibilitat del codi. Aquesta metodologia sol ser més estructurada i orientada a l'objectiu, facilitant la comprensió del codi per part dels desenvolupadors. Per altra banda, el CSS d'utilitats fa servir classes petites i específiques per aplicar estils concrets, prioritzant la flexibilitat i la rapidesa en el desenvolupament. Això sol alleugerar els fitxers CSS, ja que són més petits, tot i que pot complicar la llegibilitat en casos de llistes llargues de classes aplicades als nodes d'HTML.

Amb el CSS d'utilitats, es pot treballar de manera més àgil i iterativa, ja que es poden aplicar estils de manera més ràpida i específica. A l'hora de desenvolupar és més senzill, ja que gràcies a les classes 'atòmiques' podem imaginar com volem representar un element i posar-ho al mateix codi, sense haver de definir cap CSS addicional. Al codi podem observar que les classes dels nodes d'HTML creixen, però si miram el codi CSS s'ha reduït dràsticament, per no dir quasi al 100%.

Quines diferències has trobat entre usar una llibreria de components i una llibreria d'utilitats?

Amb la llibreria de components (Bootstrap en la PAC2) utilitzam elements predefinits, que ens abstreuen de la seva configuració, i que proporcionen una consistència visual i reutilització en tot el projecte (facilitant el manteniment, ja que els desenvolupadors poden utilitzar components predefinits assegurant una interfície d'usuari uniforme). No obstant això, pot resultar menys flexible i més pesada, ja que els components poden incorporar funcionalitats que no sempre són necessàries en el nostre desenvolupament.

Per altra banda, amb una llibreria d'utilitats utilitzam petites classes específiques als elements HTML per estilitzar-los, fent un desenvolupament més àgil i, permetent ajustar l'estil de manera ràpida i específica sense la necessitat de components predefinits. Però aquesta flexibilitat pot ser un arma de doble fil, ja que ens pot fer definir una interfície menys consistent si no es segueixen pràctiques de codificació coherents, i pot requerir una major atenció per mantenir la uniformitat visual a través del projecte.

Quines classes i components has decidit extreure i per què?

Veure apartat Ús de templates amb PostHTML i apartat Reutilització d'estils.