Rétrospective

Composants

Rétrospective

- ✓ Projet parcel vide
- ✓ Squelette HTML
- ✓ Styles CSS structurels
- ✓ Icônes
- ✓ Routeur pour les pages web
- ✓ Client pour l’API JSON
- ✓ Lecteur audio
- ✓ Local storage pour les favoris
- ✓ Détection online/offline
- ✓ Manifest PWA
- ✓ Caching
- ✓ Service worker

Projet Parcel

Rétrospective

- Il est important de se rappeler que Webpack est un “packager” qui sert à grouper plusieurs fichiers ensemble

- Il permet également de résoudre des dépendances extérieurs (par exemple, installer un package pour l’utiliser dans l’application)

- Parcel fournit un serveur web de développement pour faciliter le développement

Projet Parcel

Rétrospective

- Deux commandes principales sont utilisées pour gérer le projet

- `npm install` - Installe toutes les dépendances du projet. S'utilise typiquement à l'installation du projet ou lorsqu'une dépendance est mise à jour

- `npm run start` - Démarre le serveur web de notre projet

Markup HTML et styles CSS

Rétrospective

- Il est important d'utiliser des tags sémantiques variés pour bien différencier les éléments de l'application (privilégier <main>, <section>, <article> à)

- Pour le CSS, privilégier des classes de contrôles à appliquer à un élément, plutôt que de modifier son CSS à la main en javascript
  Par ex: la classe "active" que nous utilisons pour afficher les sections

- A noter que ces classes "active" sont manuellement définie dans le CSS que nous utilisons ! Il ne s'agit pas d'une fonction du navigateur !

Markup HTML et styles CSS

Rétrospective

- Exemple de classe de contrôle:

```css
section {
  display: none;
}

section.active {
  display: flex;
}
```

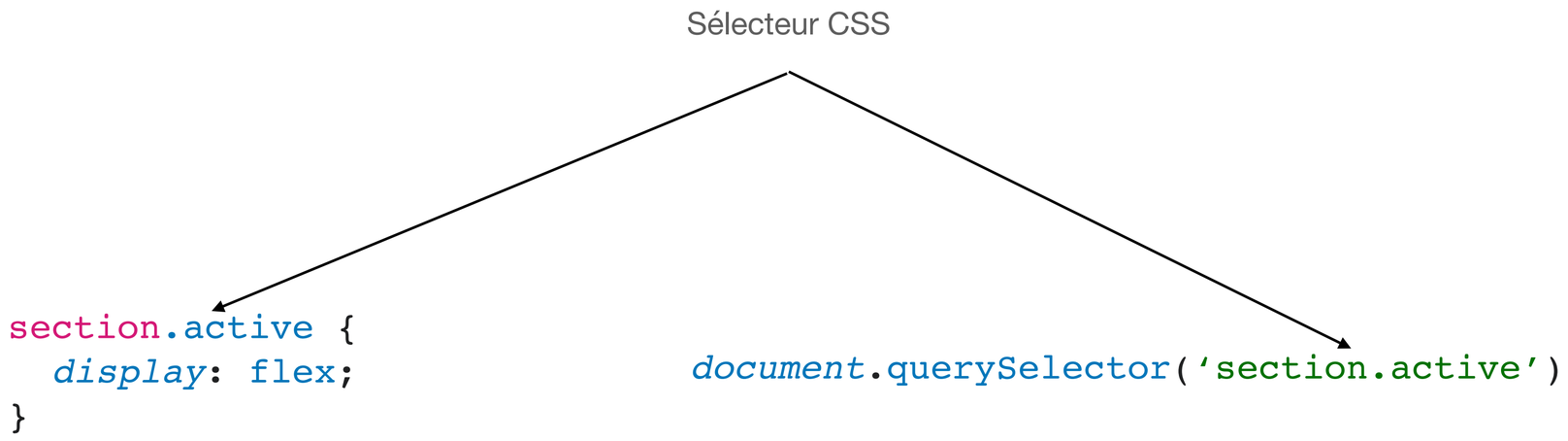
```html
<section id="ma-section">
  …
</section>
```

```js
const section = document.querySelector('#ma-section')
section.classList.add('active')
```

Markup HTML et styles CSS

Rétrospective

- Pour rappel, la chaine de caractères passée à querySelector est une chaine de sélecteur CSS



Markup HTML et styles CSS

Rétrospective

- Rappel sur l’utilisation des sélecteurs CSS…

```
section {
  …
}
```

```
<section>
  …
</section>
```

```
.section {
  …
}
```

```
<… class="section">
  …
</…>
```

```
#section {
  …
}
```

```
<… id="section">
  …
</…>
```

Markup HTML et styles CSS

Rétrospective

Un espace entre deux sélecteurs veut dire “enfant de”

```
section.active {
  …
}
```

```
<section class=“active”>
  …
</section>
```

```
section  .active {
  …
}
```

```
<section>
  <… class=“active”>
    …
  </…>
</section>
```

```
section  div.active {
  …
}
```

```
<section>
  <div class=“active”>
    …
  </div>
</section>
```

Markup HTML et styles CSS

Rétrospective

```
section.active#section-songs {
  …
}
```

```
<section class="active" id="section-songs">
  …
</section>
```

```
section  .active#section-songs {
  …
}
```

```
<section>
  <… class="active" id="section-songs">
    …
  </…>
</section>
```

```
section  .active  #section-songs {
  …
}
```

```
<section>
  <div class="active">
    <div id="section-songs">
      …
    </div>
  </div>
</section>
```

Markup HTML et styles CSS

Rétrospective

Il est possible de cumuler les sélecteurs pour le même élément

```
section.active.en-bleu {
  …
}
```

```
<section class="active en-bleu">
  …
</section>
```

```
section.active  .en-bleu {
  …
}
```

```
<section class="active">
  <… class="en-bleu">
    …
  </…>
</section>
```

```
section.active  div.en-bleu {
  …
}
```

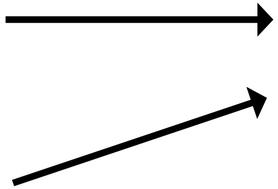
```
<section class="active">
  <div class="en-bleu">
    …
  </div>
</section>
```

Markup HTML et styles CSS

Rétrospective

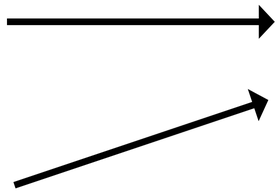
L’ordre des ids et classes n’a pas d’importance…

```
section.active.en-bleu {
  …
}

section.en-bleu.active {
  …
}
```



```
<section class=“active en-bleu”>
  …
</section>
```

```
section#section-songs.active {
  …
}

section.active#section-songs {
  …
}
```



```
<section id=“section-songs” class=“active”>
  …
</section>
```

Markup HTML et styles CSS - Icônes

Rétrospective

- Pour rappel, nous utilisons Google Material icons
- Intégré en mode CDN, via `<link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Icons" />`
- https://fonts.google.com/icons
- Exemple : `<span class="material-icons">face</span>`

Javascript - Fonctions

Rétrospective

- Rappel sur les fonctions…

```
// Déclaration d'une fonction
function maFonction() {
  …
}

// Appel d'une fonction

maFonction()

// Référence à la fonction (mais pas appelée)
// Une référence a une fonction s'utilise typiquement pour être passée en paramètre
maFonction
```

Javascript - Fonctions

Rétrospective

```
// Déclaration d’une fonction
function maFonction() {
  …
}

// Ici, on passe la référence de la fonction au listener, comme une manière de lui dire
// “quand l’événement se passe, il faut appeler la fonction “maFonction”
window.addEventListener('hashchange', maFonction)
```

Javascript - Fonctions

Rétrospective

```
// HERE BE DRAGONS
// Ici, on ne passe pas la référence à la fonction, mais on l'appelle et c'est le résultat de
// maFonction() qui sera passé comme argument au listener…
window.addEventListener('hashchange', maFonction())


// Le code ci-dessus est sémantiquement équivalent à cela:
const temp = maFonction()
window.addEventListener('hashchange', temp)
```

Javascript - Fonctions fléchées

Rétrospective

- Pour simplifier, une fonction fléchée est une fonction anonyme. En gros, une fonction qui n’a pas de nom…

- Elles sont typiquement utilisées pour être passées en argument quelque part. Par exemple, lorsque l’on a pas besoin de la réutiliser, mais qu’on est obligé de passer une fonction

- Création “on the fly” en gros…

Javascript - Fonctions fléchées

Rétrospective

```
// Dans ce cas, on peut utiliser une fonction fléchée. On n’aura certainement jamais besoin de rappeler
// cette fonction, mais nous n’avons pas le choix d’en passer une, car addEventListner nous l’impose

window.addEventListener('hashchange', () => {
  console.log(‘hello')
})
```

Javascript - Fonctions fléchées

Rétrospective

```
// HERE BE DRAGONS
// Cf. Exemple précédent sur les appels, ceci ne marche pas ! On se retrouve à nouveau dans le cas
// précédent où console.log sera directement appelé et c’est son résultat qui sera utilisé comme
// paramètre. De l’intérêt de le mettre dans une fonction fléchée

window.addEventListener('hashchange', console.log(‘hello'))
```