Ludwig GUERIN

Info 2 Groupe 2

Mini projet Web Client Riche - Flickr API

Architecture

```
.
|_ dev
|   |_ js
|   |   |_ setup
|   |   |   |_ setupAutocomplete.js
|   |   |   |_ setupForm.js
|   |   |   |_ setupImage.js
|   |   |   |_ setupTabs.js
|   |   |
|   |   |_ index.js (point d'entrée)
|   |   |_ loadJquery.js
|   |   |_ prototypes.js
|   |   |_ urls.js
|   |
|   |_ sql
|       |_ db.sql (pour mise en place de la base de données)
|
|_ node_modules
|   |_ [...]
|
|_ public_html
|   |_ assets
|   |   |_ css
|   |   |   |_ main.css (styles de la page)
|   |   |   |_ [...]
|   |   |
|   |   |_ fonts
|   |   |   |_ [...]
|   |   |_ images
|   |   |   |_ [...]
|   |   |_ js
|   |   |   |_ index.bundle.js (fichier compilé)
|   |   |
|   |   |_ js-libs (bibliothèques récalcitrantes)
|   |   |   |_ dataTables.min.js
|   |   |   |_ fancybox.min.js
|   |   |   |_ row-grid.min.js
|   |   |
|   |   |_ json
```

```
|   |       |_ autocomplete.json (configuration de l'autocomplete)
|   |       |_ config.json (configuration de l'application)
|   |       |_ flickr.json (configuration de l'utilisation de l'API)
|   |
|   |_ commune.php
|   |_ index.html
|
|_ [...]
```

Configurations (dans `./public_html/assets/json/`)

autocomplete.json

`root` défini l'URL (absolue ou relative) du fichier PHP à utiliser pour l'autocomplete (fichier donné ou URL proposée dans le sujet).

NB:

Si l'URL indiquée est relative, cela doit être par rapport au fichier `index.html`.

eg. `/projet/public_html/script.php` --> `script.php`

config.json

- `form` : Sélecteur CSS pour le formulaire de recherche
- `autocomplete` :
  - `container` : Sélecteur vers le champs nécessitant l'autocomplete
  - `minLength` : Nombre minimal de caractères pour lancer l'autocomplete
  - `maxRows` : Nombre maximal d'éléments à récupérer par l'autocomplete
- `tabs` :
  - `container` : Sélecteur vers le conteneur des onglets (utilisé pour jQuery UI)
  - `grid` : Sélecteur vers le conteneur pour la vue grille
  - `table` : Sélecteur vers le conteneur pour la vue table
- `rawTableID` : ID à donner à la table de la vue table
- `rawTableSelector` : Sélecteur vers la table de la vue table
- `galleries` :
  - `grid` : Nom de la galerie pour les images de la vue grille
  - `table` : Nom de la galerie pour les images de la vue table
- `headers` : En-têtes de la table de la vue table

flickr.json

- `key` : Clé d'API
- `root` : URL de communication vers l'API

NPM Scripts

`npm run setup` permet l'installation et la MàJ des dépendences.

`npm run build` permet le bundling de l'application

Webpack

Bien que le fichier de configuration (`webpack.config.js`) soit suffisamment compartimenté, je vais l'expliciter ici.

`@js` est un alias pointant vers le dossier `./dev/js/`.

La configuration a été adaptée à l'utilisation de `webpack-jquery-ui` (loaders pour les images et les styles respectifs).

Bibliothèques

Au-delà des dépendances de développement pour webpack, les bibliothèques suivantes ont été utilisées :

- `jquery`
- `jquery-ui` (par le biais de `webpack-jquery-ui`)
- `datatables` (pour la vue table)
- `rowgrid.js` (pour la vue grille)
- `jq-flash` (pour les messages d'erreur à l'intention de l'utilisateur)

Remarques

Bien qu'apparues après leur équivalent jQuery, les `Promise` ont l'avantage d'être des monads :

```
monad :: (Monad<T>, T -> (Monad<E> | E) -> Functor<E>
```

Là où les `$.deferred` semblent n'être que des foncteurs :

```
foncteur :: (Functor<T>, T -> E) -> Functor<E>
```

L'utilisation de `Promise` via `fetch` aurait pu être plus profitable.

```
Promise.all([
    fetch("a"),
    fetch("b"),
    fetch("c")
]).then(([a, b, c]) => fetch(`deb/${a}/${b}/${c}`))
.then(console.log)
.catch(console.error);

//VS

$.when($.get("a"), $.get("b"), $.get("c"))
.done(([a], [b], [c]) => {
    $.get(`deb/${a}/${b}/${c}`)
    .done(console.log)
    .fail(console.error);
}).fail(console.error);
```