Front-End : de 1990 à 2010

L. Daverio et O. Hermant

12 décembre 2022

Ce TP a pour but de vous faire (re-)voir des concepts que nous utiliserons intensivement dans la programmation mobile :

- les balises et les attributs,
- le style CSS,
- le langage javascript,
- le modèle de programmation client-serveur à travers une API, vu du côté client,
- la programmation asynchrone avec des fonctions de callback.

La serveur de ce modèle de programmation seront abordées plus tard.

1 Pages web au XXe siècle : 100% HTML

Exercices :

- créer une page web vide (vous pouvez tout à fait la copier-coller des exemples de cours), et la visualiser dans un navigateur (`Open File (Ctrl + O)`)
- ajouter à votre page une division (balise `<div>` – ne pas oublier la balise fermante) et du texte à l'intérieur
- dans cette div, ajouter une liste non ordonnée (balise `<ul>`), avec quelques items (`<li>`)
- ajouter un peu de style : des bords noirs à la div précédente
  - il faut définir l'attribut `style` à la div
  - la propriété de style à définir est `border`, que vous pouvez fixer à `solid`
- faites en sorte que les éléments de la liste commence par une cercle blanc (ou tout autre style de votre choix) au lieu d'une puce,
  - comme
  - ceci.

- (facultatif) ajouter encore plus de style (bords de couleur bleue, texte aligné à droite, couleur de fond de la div, etc). Voir par exemple

  `https://www.w3schools.com/cssref/`

  pour une référence complète sur les propriétés de style.

- faites valider chacune de vos pages précédentes :

  `https://validator.w3.org`

  Ceci est important car les navigateurs ne sont pas stricts et acceptent facilement du HTML mal formé, en essayant d'en faire quelque chose malgré tout.

2 Les formulaires

Les formulaires permettent d'envoyer des requêtes à un serveur. Un formulaire est inclus dans la balise suivante

```
<form action="https:/kiva.mobapp.mines-paristech.fr/api/trace" method="XXX">
</form>
```

qui comporte deux attributs

- `action`, qui est l'URL de la requête ;
- `method` qui est le type HTTP de la requête (`GET` ou `POST` dans le cas des formulaires).

Un formulaire contient un certain nombre de champs. Nous allons utiliser un champ d'entrée de type texte, qui est une balise autofermante.

```
<input type="text" name="param" value="valeur"/>
```

Outre l'attribut `type`, chaque champ d'entrée doit avoir un nom (attribut `name`), qui sera passé à la requête HTTP en même temps que sa valeur. L'attribut `value` est optionnel et contient la valeur par défaut du champ d'entrée.

Chaque balise `<input>` fournit un paramètre à la requête qui est faite au serveur lors de la soumission du formulaire. Ce passage de paramètre se fait soit sous une forme optionnelle (i.e., dans la barre d'adresse), dans le cas de la méthode GET ; soit dans un dictionnaire de paramètres, dans le cas de la méthode POST. Plus de détails sur ce sujet lors des cours sur la conception du serveur.

La soumission d'un formulaire se fait avec un champ particulier : un bouton de soumission.

```
<button type="submit" value="submit">Envoyer</button>
```

Exercices :

- implémenter une formulaire (très) basique, qui soumettra une requête à l'adresse `https:/kiva.mobapp.mines-paristech.fr/api/trace`. Tester les deux méthodes `GET` et `POST`.
- ajouter au moins un champ d'entrée de type texte : lors de la soumission du nouveau formulaire, l'adresse de la requête a-t-elle changé ? (attention, ceci est une question piège). La réponse du serveur a-t-elle changé ?

3 L'API KiVa

`https://kiva.mobapp.mines-paristech.fr` est un serveur que nous apprendrons à développer début 2023. Cette année, nous nous contenterons de l'exploiter du côté "client".

Ce serveur implémente quelques opérations de manipulation d'une table d'associations clefs-valeurs (stockée dans une base de données) qui contient par défaut trois couples :

clef	valeur
calvin	hobbes
hello	world
kiva	cool

À partir de l'adresse de base `https://kiva.mobapp.mines-paristech.fr`, il est possible de consulter cette table par une requête de type `GET` sur la "route"[1] `/api/store`.

Pour aller vérifier cela en pratique, cela signifie qu'il vous suffit de taper dans la barre d'adresse : `https://kiva.mobapp.mines-paristech.fr/api/store`

Toute l'api (sauf `/api/trace` et `/api/version`) est protégée par un mot de passe, il s'agit du login `"kiva"` et du mot de passe `"bien"`.

Les routes accessibles sur cette API sont les suivantes :

Requête client			Réponse serveur		
Méthode	Route	Paramètres	Contenu	Format	Type
`GET`	`/api/store`	—	Liste clefs/valeurs	JSON	200
`GET`	`/api/store`	`filter`	Liste clefs/valeurs filtrée sur les clefs selon la valeur de `filter`	JSON	200
`GET`	`/api/store/<key>`	—	Valeur associée à `<key>`	JSON	200
`POST`	`/api/store`	`key`, `val`	Ajoute une nouvelle paire	—	201
`PUT`	`/api/store/<key>`	`val`	Modifie la paire associée à `<key>`	—	201
`PATCH`	`/api/store/<key>`	`val`	Modifie la paire associée à `<key>`	—	201
`DELETE`	`/api/store/<key>`	—	Supprime la paire associée à `<key>`	—	204

Deux exemples pour mieux comprendre, en supposant avoir le contenu original de la base de données :

- la requête `GET https://kiva.mobapp.mines-paristech.fr/api/store/calvin` renverra `"hobbes"`

[1] ce terme sera explicité l'année prochaine

- la requête `GET https://kiva.mobapp.mines-paristech.fr/api/store?filter=%25l%25` renverra `[["calvin","hobbes"],["hello","world"]]`, car le filtrage aura été effectué en SQL par `where key like %l%` (l'encodage du caractère `%` se fait, dans la barre d'adresse, avec `%25`).

Exercices :

- essayez à la main (i.e. dans la barre d'adresse de votre navigateur) ces requêtes, ainsi que quelques autres.
- implémentez un formulaire pour la première requête `GET`.
- modifiez ce formulaire pour la deuxième requête `GET` (celle avec le paramètre filter), en ajoutant un champ d'entrée `input` bien choisi.
  Assignez à ce champ d'entrée une valeur par défaut qui permette de sélectionner tous les couples.
- implémentez (séparément) un formulaire pour la requête `POST`.

La soumission de formulaire ne permet pas de faire des requêtes par une méthode autre que `POST` ou `GET` et encore, la troisième requête `GET` est pour l'instant inaccessible.

De plus, le retour est un affichage en mode texte d'informations au format JSON. C'est un peu rudimentaire. Pour ces raisons, nous allons rendre les pages web *dynamiques*.

4 Pages dynamiques avec Javascript

Javascript permet d'exécuter du code du côté du client, en l'occurrence dans votre navigateur. Ce dernier l'exécute lorsqu'il le rencontre (au chargement de la page), ou en réaction à un événement. C'est cette dernière manière qui va nous intéresser, elle exploite l'attribut `onClick` de la balise <**button**>.

Exercices :

- reprenez (dans un nouveau fichier html) votre/vos formulaire de la page précédente, et supprimez les balises `<form>`. Nous ne soumettrons plus les requêtes au serveur par le biais des formulaires.
- faites réagir le bouton de soumission à l'événement `onClick`, afin de lui faire exécuter le code `alert('Hello, World')`. Testez.
- on veut maintenant écrire du texte dans la page plutôt que d'avoir un popup. Pour ceci, il faut :
  1. une div, avec un attribut id (pour identifiant), qui a une valeur de votre choix, mais doit être unique ;

2. en JS, accéder à cette div par le DOM, en appelant la fonction `querySelector()` sur l'objet `document` – fonction à laquelle on doit fournir l'identifiant de l'élément recherché précédé d'un `#`. Exemple, si on veut récupérer la `<div id="cestmoi">`, il faudra la rechercher comme ceci : `document.querySelector('#cestmoi')`.[2]

   Cette objet a, entre autres, un attribut (`innerHTML`) que l'on peut modifier, par exemple `document.querySelector('#cestmoi').innerHTML = "pnom"`. On peut aussi, en deux lignes, procéder ainsi :

```
const maDiv = document.querySelector('#cestmoi')
maDiv.innerHTML = "pnom"
```

Testez.

- on veut maintenant incrémenter un compteur et en afficher la valeur dans la div ci-dessus. Il faut pour cela déclarer une variable, ce qui n'est pas possible dans l'attribut `onClick` de notre bouton.

  Vous pourrez initialiser ce compteur dans l'entête (`<head>`) du fichier html, à l'intérieur d'une balise

```
<script type="text/javascript">
</script>
```

  Ce morceau de script sera exécuté au chargement de la page.

- recupérez la valeur du champ d'input (si vous avez supprimé ce champ, rajoutez-le) lors du clic, et affichez-le dans la div. Pour un élément du DOM (≈ une balise) de la catégorie `input`, la valeur du champ est située dans l'attribut `value`. Si votre champ d'input a comme id `"monChamp"`, alors cela se fait typiquement comme ceci:

```
document.querySelector('#monChamp').value
```

- pour limiter la taille du code écrit dans l'attribut `onClick`, mettez-le dans une fonction séparée, que vous mettrez dans la balise `<script>` ci-dessus.

- encore mieux, faites un fichier `code.js` séparé, qui vous inclurez avec l'attribut `src` de la balise `script`.

5 Tableau de Conversion Celsius-Fahrenheit

On rappelle les formules de conversion des degrés Celsius en degrés Fahrenheit :

$$X°\mathbf{C} = \frac{9}{5} * X + 32)°\mathbf{F}$$

[2] Cette manière de faire nous suffira amplement, mais on peut chercher un élément par de nombreuses manières. Voir la documentation en ligne de `querySelector` pour plus de détails.

Affichez (dans votre page html) la table de conversion des degrés Celsius (de 0 à 100) en Fahrenheit.

Indications :

- un tableau, en HTML, se déclare comme un élément <**table**>, qui contient des <**tr**> (lignes) qui, elles-mêmes contiennent des <**td**> (cellules). Ainsi :

```
<table id="fibleaunacci">
<tr> <td>2</td> <td>3</td> <td>5</td> </tr>
<tr> <td>8</td> <td>13</td> <td>21</td> </tr>
</table>
```

  sera rendu ainsi

2	3	5
8	13	21

- la syntaxe la plus simple pour les boucles en JS est
  `for(let i=0 ; i<=100; i++){ /* du code */ }`

Exercice facultatif : faire, en plus, une calculette "à la volée" dans le style de `https://www.google.com/search?q=celsius+to+fahrenheit` :

- lancez la conversion à chaque fois qu'un bouton (de votre choix) est cliqué.
- (plus difficile) la même chose, mais sans bouton. Vous pourrez vous intéresser aux attributs/événements `onChange` et `onFocus` (voire à des événements encore plus précis).

6 Requêtes HTTP en javascript (s'il reste du temps)

6.1 Principe

Plutôt que de visualiser la réponse du serveur à notre requête, on cherche maintenant à *générer* une page HTML avec ces informations.

1. Solution 1 : côté serveur. L'idée est de serveur de renvoyer (éventuellement en la générant à la volée) une jolie page. Bienvenus dans la programmation web du début des années 2000. (on fait un peu mieux depuis avec les templates, etc).
2. Solution 2 : côté client. L'idée est de demander à javascript de faire la requête, puis de *modifier* les éléments de la page courante, grâce au DOM et aux informations reçues, ainsi qu'à des librairies bien choisies (e.g. jQuery, Node, Angular, etc).

Nous allons explorer la solution 2. Elle est par nature **asynchrone**, c'est à dire que l'utilisateur clique sur un bouton, puis il vacque à ses occupations sans attendre la réponse du serveur kiva. Ainsi, le code JS exécuté lors du clic devra-t-il prévoir l'exécution d'une fonction *après que* le serveur aura répondu à la requête, et il rendra la main sans attendre.

Enregistrer une telle fonction de réponse (qui s'appelle *callback*) au moment de l'envoi de la requête est typique de la programmation web AJAX (Asynchronous Javascript And XML).

6.2 Préliminaires

À faire :

- Nous suggérons de mettre toutes vos fonctions dans l'en-tête, à l'intérieur d'une balise <script> (ou dans un fichier .js), et de ne mettre que le minimum de code JS dans la propriété onClick des boutons, voire pas du tout (cf. le cours).
- Écrire une fonction `function update(data)` qui prend des données (sous forme textuelle) et les écrit dans une <**div**>. Affecter cette fonction au clic sur un bouton, et la tester.

6.3 Mise à jour de la page par des requêtes au serveur

Importer, dans le <**head**> de votre page, la librairie suivante : `simplerequest.js`. Elle contient une unique fonction, `req` qui prend les paramètres suivants :

- la méthode (`"GET"`, `"POST"`, ...)
- l'adresse de base (exemple : `"https://kiva.mobapp.mines-paristech.fr"`)
- la route à laquelle on souhaite soumettre la requête sur cette adresse (exemple : `"/api/store"`)
- le nom et le mot de passe, pour l'authentification
- les paramètres de la requête (si nécessaire), sous forme d'un objet javascript (POJO), par exemple `{"filter" : "cal\%"}`
- la fonction de callback, qui sera appelé lorsque la requête aura répondu favorablement.

Par exemple, un appel à `req('GET','https://kiva.mobapp.mines-paristech.fr', '/api/store', 'kiva', 'bien', {}, update)`, appellera la fonction `update()`, en lui fournissant les données récupérées par la requête au serveur, une fois que celles-ci seront reçues.

Cela aura pour effet, in fine, de mettre à jour la <**div**> de la section 6.2 ci-dessus avec les données reçues (i.e., la liste de toutes les clefs-valeurs).

Exercices :

- implémentez toutes les requêtes correspondant aux fonctionalités de l'API
  - avec des champs <**input**> permettant de donner des valeurs pour les requêtes qui envoient des paramètres au serveur.
  - Formatter correctement (en HTML) le JSON reçu. Par exemple si le serveur renvoie une liste, afficher cette dernière dans un tableau HTML. Puis ajouter du style.
  - Pour la requête `GET` avec paramètre `filter`, faire en sorte que, si le champ <**input**> correspondant est vide, la requête soumise soit faite *sans* le paramètre filter (plutôt qu'un paramètre vide).
  - nettoyez l'ensemble des clefs-valeurs que vos camarades ont ajouté depuis le début du TP.

7 Pour la fin : authentification par token (facultatif)

S'il vous reste encore du temps.

- Lisez attentivement le code de `simplerequest.js`, et au lieu d'utiliser la fonction `req`, faites la requête `POST` "à la main" avec `fetch`.
- passez à de l'authentification par token, qui est une manière plus standard de faire. Cela sera utilisé plus tard lors de la partie "développement mobile" :

  1. On commence par faire une première requête `GET` sur la route `/api/login`
  2. le serveur renvoie un *token* (jeton) que l'on passe ensuite dans l'en-tête de toutes les requêtes suivantes. C'est à dire que, au lieu d'ajouter aux requêtes l'en-tête `"Authorization : Basic "` (plus le hachage du login et du mot de passe), il faut ajouter l'en-tête `"Authorization : Bearer "` (plus le token reçu).

     Le jeton en soi est la dernière partie de la chaîne de caractère renvoyée par la première requête.

  L'authentification n'a ainsi lieu que lors de la première requête. Au moins jusqu'à ce que le token expire.
- Programmez une mise à jour régulière et automatique de la liste clefs-valeurs (avec la fonction javascript `setInterval()`)
- Ajoutez, à côté de chacune des entrées de cette liste (dans un tableau), un bouton `Delete` qui permet de supprimer exactement cette entrée.

8 Ressources supplémentaires

Le site `https://www.w3schools.com` contient suffisamment de ressources sur tous les thèmes abordés dans ce TP et dans ce cours. Il en existe bien entendu une multitude d'autres, telles que openclassrooms, la documentation des API des librairies, etc.