📅Projet : Affichage du calendrier

Slides :

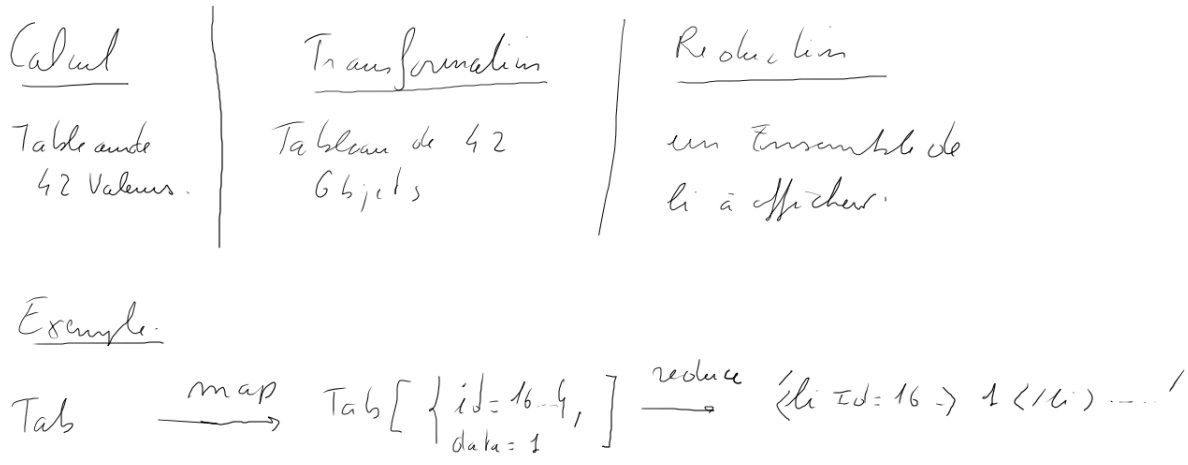
🟦Calendar

Présentation

🚀Imaginez que vous présentiez votre projet à des collaborateurs. Quel serait votre discours ?

🆒Si je ne dispose que d'une minute pour faire passer mon message, je résumerais le projet avec un seul transparent !


En voici une ébauche :



🔧La partie haute du schéma montre l'idée générale avec un vocabulaire peu technique.

Calcul : une équipe génère un tableau correspondant à un mois à afficher.

Transformation : une équipe intermédiaire transforme ce tableau en un tableau d'objets.

Réduction : une équipe réduit le tableau transformé en une liste d'éléments HTML à afficher.

🔧La partie base du schéma "Exemple" précise les valeurs du tableau à l'issue de chaque étape.

tab de 42 valeurs en milliseconde		tab de 42 objets avec deux propriétés		string de 42 avec un id

`[…`
`new Date(1604098800000),`
`new Date(1604185200000),`
`…]`
	m
	a
	p
		`[…`
		`Object {`
		`  data: 31,`
		`  id: 1604098800000}`
		`Object {`
		`  data: 1,`
		`  id: 1604185200000}`
		`…]`
			r
			e
			d
			u
			c
			e
				`` `… ``
				`<li id=1604098800000>31</li>`
				`<li id=1604185200000>1</li>`
				`…` ``

🔧Techniques

Nous allons faire le point sur les notions du langage que nous devons maîtriser pour réaliser notre projet.

Reprenons tout d'abord les trois étapes précédentes

1. **Calcul**
2. **Transformation**

3. **Réduction**

1️⃣Calcul

La partie Calcul utile principalement l'objet Date pour générer les dates des 42 jours à afficher. On obtient un tableau du type :

const cal = [..., new Date(1604098800000),new Date(1604185200000), ...];

2️⃣Transformation

Sans surprise, l'étape de transformation utilise la méthode map. Voici un exemple de base :

```
1. const trans = cal.map( function (oneDate) {
2.         return {
3.             id: oneDate.getTime(),
4.             data : oneDate.getDate()
5.         }
6. });
```

3️⃣Reduce

Pour obtenir finalement le string à afficher, la méthode reduce est utilisée.

🪛Exemple :

Voici un exemple de base pour obtenir un string contenant les balises <li> à afficher.

```
1. const affichage = trans.reduce((acc, item) =>
2.           acc + `<li>${item.data}</li>`, '');
```

test

🥷Template

Nous allons aborder une nouvelle notion : le Template[1].

💡**Un template** définit un motif de base que l'on doit dupliquer un grand nombre de fois.

🪚Revenons à la partie calcul qui génère un tableau de valeurs de type objet.

Revoici la partie  calcul[2] qui génère un tableau de valeurs.

```
1.  function calendarT(month, year) {
2.      const  OFFSET_MONTH = new Map([
3.              [1, 6], // Monday
4.              [2, 0],
5.              [3, 1],
6.              [4, 2],
7.              [5, 3],
8.              [6, 4],
9.              [0, 5] // Sunday
10.         ]);
11.
12.     let firstOfMonth = new Date(year, month, 1),
13.         dayFirst = firstOfMonth.getDay(),// Sunday - Saturday : 0 - 6
14.         dec = OFFSET_MONTH.get(dayFirst),
15.         cal = [];
16.
17.     for (let i = 0; i < 42; i++) {
18.         let d = new Date(year, month, i - dec);
19.         cal.push(d);
20.     }
```

[1] Nous pourrons dans un second temps définir une classe (lien).
[2] python

```
21.    return cal;
22.}
```

Nous voulons afficher les valeurs de ce tableau à l'aide d'un template.

Template de base

Pour découvrir la mécanique des templates, nous allons prendre un exemple simple.

🔧L'idée est d'afficher à l'aide d'une grille (7,6) les valeurs d'une liste.

Le template représente le motif de base à dupliquer ; c'est-à-dire un élément de la liste.

Comme un élément d'une liste <ul> en HTML est la balise <li>, nous définissons simplement le template par :

```
const defaultTemplate = `<li>{{data}}</li>`
```

Nous allons comprendre le rôle des {{}} dans très peu de temps. La valeur data joue le role de variable. Il faudra la remplacer dans le template.

Finalement, pour remplir notre liste, il nous faut dupliquer ce template autant de fois qu'il y a de valeurs puis remplacer la variable.

🪛Pour ce faire, nous pouvons itérer sur les valeurs du tableau comme suit :

```
1.  function show(tableau) {
2.      let view = '';
3.
4.      for (let data of tableau) {
5.
```

```
6.          let template = defaultTemplate;
7.
8.          template = template.replace('{{data}}', data.getDate());
9.
10.         view = view + template
11.     }
12.
13.     return view
14.}
```

Lig.2 : On initialise la vue à vide

Lig.6 : On duplique le template

Lig.8 : {{data}} va être remplacée par la valeur du jour de la date. Pour cela, nous utilisons simplement la méthode **replace**[3].

Lig.10, la concaténation accumule les valeurs de la liste dans un string au départ vide.

Finalement, nous transformons notre tableau de 42 objets en un string de 42 balises <li>.

Voici le code résultant :

'
28
29
30
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
1
2
3
4
5
6
7
8
'

Il ne reste plus qu'à ajouter cette liste dans notre page HTML.

La valeur à ajouter n'est pas un élément au sens du DOM, c'est un String représentant du HTML.

[3] replace mdn

Nous utilisons la méthode **insertAdjacentHTML**[4].

```
ul.insertAdjacentHTML('afterBegin', show(calendarT()));
```

code

Projet

Nous allons répondre à différentes questions !

🚀Nous allons tenter de comprendre l'importance de mettre en place un processus robuste et réutilisable pour répondre aux différentes contraintes.

Mise en avant du jour courant

🪛Nous voudrions mettre en avant le jour courant[5].

28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

[4] insertAdjacent
[5] Dans la figure suivante, 22 est le jour courant du mois affiché.

🔧Il existe bon nombre de solutions à ce problème, nous allons nous focaliser sur une idée simple avec une utilisation du CSS.

🪛Si nous pouvions inspecter le code de la solution, nous aurions ce type de résultat :

Nous voyons dans la figure précédente que les <li> disposent d'un attribut id et que seul le jour courant (ici le 8) possède une class today.

🪛Trouvez le template à définir.

Dans l'exemple précédent le template de base était

```
const defaultTemplate = `<li>{{data}}</li>`
```

Pour obtenir le HTML vue précédemment lors de l'inspection, il nous faut modifier le template comme ceci :

```
const defaultTemplate = `<li id=time{{id}} > {{day}} </li>`
```

Maintenant, si nous reprenons les trois étapes de notre processus

1. **Calcul**
2. **Transformation**
3. **Réduction**

La partie 1 calcul reste inchangée. Il nous reste à définir les deux dernières étapes.

2 Transformation

Dans un premier temps, nous transformons le tableau de valeurs de notre calendrier en un tableau d'objets de type.

```
{
   id: date.getTime(),
   data : date.getDate()
}
```

La transformation du tableau est obtenue par l'utilisation de la méthode **map**[6].

```
function mapCalendar(tab) {

      let mapToObjet = function (date) {
      return {
         id: date.getTime(),
         data : date.getDate()
      }
   }

   return tab.map(mapToObjet);

}
```

lig. 4-7 : Chaque objet date du tableau est transformé en un objet

3 réduction

[6] map

La réduction du tableau d'objets est réalisée à l'aide d'une fonction suivante.

show = function (calT) {

```
1.      let view = '';
2.
3.      for (let data of calT) {
4.          let template = defaultTemplate;
5.
6.          template = template.replace('{{day}}', data.data    );
7.          template = template.replace('{{id}}', data.id);
8.          view = view + template
9.      }
10.
11.     return view;
12.}
```

Lig. 4 : On duplique le template.

Lig. 6-7 : On remplace les valeurs.

Lig. 8 : Chaque motif ainsi complété est concaténé à la vue.

code

🥷Reduce

Nous pouvons également utiliser la méthode **reduce** pour réduire notre tableau.

```
1.  show = function (calT) {
2.
3.    return calT.reduce( (acc, item)
4.              => acc + `<li id="${item.id}">${item.data}</li>`,``);
5.  }
```

Lig.4 : La valeur de l'accumulateur est initialisée à un string vide ``. On accumule à chaque itération une balise <li> avec un attribut id.

code

CSS

Dans la suite, c'est le grand retour du CSS. Il faut privilégier son utilisation dans bien des cas.

Ainsi, pour afficher le jour courant, nous utilisons le CSS[7] par l'intermédiaire de l'ajout d'une classe.

Voici le code qui permet de rechercher la balise à mettre en valeur

```
1. elementToday = document.querySelector(`#time${today}`);
2. elementToday.classList.add("today");
```

Lig.1 : Nous recherchons dans un premier temps la balise <li> dont l'identifiant correspond à celui d'aujourd'hui[8].

Lig.2 : Nous ajoutons la classe today à la balise correspondante.

Affichage

Le CSS peut être réduit au minimum grâce à l'utilisation d'une grille.

```
1. .cal {
2.     display : grid;
3.     width : 50vw;
4.     grid-template-columns : repeat(7, auto);
5.     grid-template-rows :  repeat(6,auto);
6.     grid-gap : 5px;
```

[7] https://developer.mozilla.org/fr/docs/Web/CSS/S%C3%A9lecteurs_d_attribut

[8] let t = new Date(),
today = new Date(t.getFullYear(), t.getMonth(), t.getDate()).getTime();

```
}

.cal li {
  display : flex;
  align-items : center;
  justify-content : center;
  list-style : none;
  border-radius: 50%;
}
.today {
  background : indianred;
  color : white;
}
```

🥷Projet

🪛Nous voudrions afficher le numéro de la semaine.

40 28	29	30	1	2	3	4
41 5	6	7	8	9	10	11
42 12	13	14	15	16	17	18
43 19	20	21	22	23	24	25
44 26	27	28	29	30	31	1
45 2	3	4	5	6	7	8

💡L'idée est de calculer pour chaque jour le numéro de la semaine.

Cette valeur stockée dans un attribut ne sera visible que pour les Lundi.

Si vous reprenez les étapes de création de notre vue. Nous définissons une 2️⃣transformation puis une 3️⃣réduction.

Le CSS permettra de n'afficher que les valeurs des semaines pour les lundi.

Nous reprenons la trame du projet précédent.

2️⃣Transformation + Trouvez le template à définir

3️⃣réduction

Nous mettrons l'accent sur le CSS

🪛Trouvez le template à définir.

Si nous inspections la solution, vous aurions le HTML suivant pour chaque

```
▼<li data-nbofweek="40">
   ::before
   "28"
  </li>
  <li data-nbofweek="40">29</li>
```

🥷Voici le template que nous définissons :

```
<li data-nbOfWeek="{{nbOfW}}"> ${{data}} </li>
```

Nous reviendrons bien évidemment sur l'intérêt de stocker la valeur du numéro de semaine dans un attribut

2️⃣Transformation

La transformation du tableau est obtenu par l'utilisation de la méthode map[9].

```
1. function mapCalendar(tab) {
2.
3.   function getNumberOfWeek2(day) {
4.     const firstDayOfYear = new Date(day.getFullYear(), 0, 1);
5.     const pastDaysOfYear = (day - firstDayOfYear) / 86400000;
6.     return Math.ceil((pastDaysOfYear + firstDayOfYear.getDay() + 1) / 7);
7.   }
8.
```

[9] map

```
9.   let mapToObjet = function (date) {
10.   return {
11.     nbOfW: getNumberOfWeek2(date),
12.     data: date.getDate()
13.   };
14. };
15.
16. return tab.map(mapToObjet);
17.}
```

Lig.3 : La fonction getNumberOfWeek2 permet de générer le numéro de la semaine.

Lig. 5 : 86400 = 24*60*60= Nombre de secondes en 24 heures.

Lig. 9-12 : Chaque date du tableau est transformée en un objet.

3réduction

La réduction se fait par la méthode reduce.

```
1. show = function (calT) {
2.   return calT.reduce( (acc, item)
3.       => acc + `<li data-nbOfWeek="${item.nbOfW}">${item.data}</li>`,
4.     ``
5.   );
6. };
```

Lig.3 : On définit un attribut de données data-nbOfWeek[10] pour stocker une valeur que l'on pourra utiliser.

[10] Ils permettent d'échanger des données propriétaire entre le HTML et la représentation du DOM, qu'on peut manipuler avec des scripts. (Lien)

CSS

La ~~magie~~ force du CSS est grande.

```
1.  .cal li:nth-child(7n+1)::before{
2.      position : relative;
3.      top : -10px;
4.      font-size : 0.5em;
5.      content : attr(data-nbOfWeek);
6.  }
```

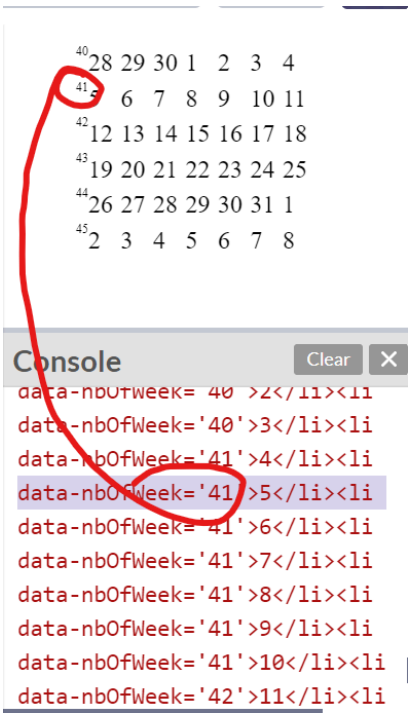
Le sélecteur lig.1 nth-child(7n+1) permet d'agir sur les <li> d'indices [1,8,15…]. Ces indices correspondent aux premiers jours des semaines : les Lundi.

n	7n+1
0	1
1	8
2	15

::before[11] permet d'injecter du contenu dans le <li> lig. 5

Le contenu injecté uniquement pour les lundi est la valeur de l'attribut nbOfWeek

[11] https://developer.mozilla.org/fr/docs/Web/CSS/::before



code

Remarque

Le code n'est jamais unique, en voici une autre version.

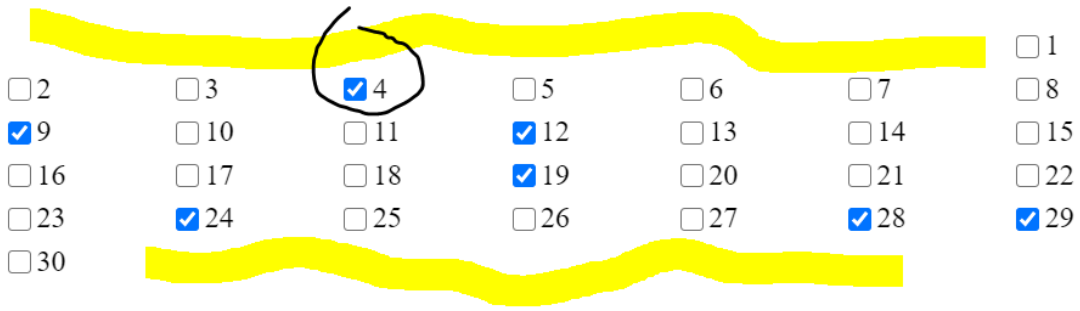
code

🪛Projet

Affichez le calendrier du mois suivant !

🪛Projet

Voici un calendrier avec simulation de RDV.



L'inspection du code donne :

```
1.  <li data-id="1604098800000" class=" notMonth">
2.          <div class="view">
3.            <input class="toggle" type="checkbox">
4.            <label>31</label>
5.          </div>
6.        </li>
```

Les jours du mois courant ont pour class **month** sinon ***notMonth***. La classe notMonth rend le jour invisible (bande en jaune).

🔧Simulation

On génère aléatoirement des coches (simulants des RDVs).

```
1.  <li data-id="1604358000000" class=" month">
2.          <div class="view">
3.            <input class="toggle" type="checkbox" checked>
4.            <label>4</label>
5.          </div>
6.        </li>
```

Template

Nous allons examiner le template et son utilisation.

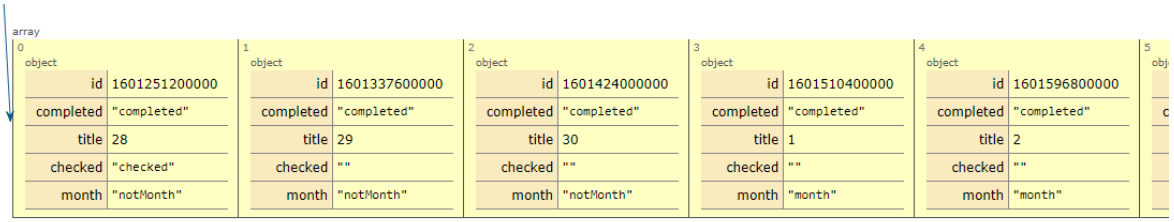
```
const defaultTemplate = `
      <li data-id="{{id}}" class="{{completed}} {{month}}">
        <div class="view">
          <input class="toggle" type="checkbox" {{checked}} />
          <label>{{title}}</label>
        </div>
      </li>
    `
```

🪛Commençons par transformer notre tableau en un tableau d'objet.

```
function mapCalendar(tab, month, year) {
    let t = new Date(),
        today = new Date(t.getFullYear(), t.getMonth(), t.getDate()).getTime();


    let mapToObjet = function (date) {

        let completed = '',
            checked = '',
            ismonth = 'notMonth';

        let currentMonth = date.getMonth();

        if (date < today) {
            completed = 'completed';
        }
        if (date == today) {
            completed = 'today';
        }

        if (currentMonth == month) {
            ismonth = 'month';
        }

        //simu RDV
        if (Math.random() > 0.8) {
            checked = "checked";
        }

        return {
            id: date.getTime(),
            completed,
            title: date.getDate(),
```

```
            checked,
            month: ismonth,
        }
    }
    return tab.map(mapToObjet)

}
```

L'idée est d'obtenir un tableau d'objets.



array

0 object		1 object		2 object		3 object		4 object		5 obj
id	1601251200000	id	1601337600000	id	1601424000000	id	1601510400000	id	1601596800000	
completed	"completed"	completed	"completed"	completed	"completed"	completed	"completed"	completed	"completed"	
title	28	title	29	title	30	title	1	title	2	
checked	"checked"	checked	""	checked	""	checked	""	checked	""	
month	"notMonth"	month	"notMonth"	month	"notMonth"	month	"month"	month	"month"	

Disposant de ce tableau d'objets, il suffit d'écrire une fonction qui remplace les valeurs dans le template.

```
show = function (data) {
    let view = '';


    for (let d of data) {
        console.log(JSON.stringify(d))
        let template = defaultTemplate;

        template = template.replace('{{id}}', d.id);
        template = template.replace('{{title}}', d.title);
        template = template.replace('{{completed}}', d.completed);
        template = template.replace('{{month}}', d.month);
        template = template.replace('{{checked}}', d.checked);

        view = view + template
    }

    return view
}
```

🔺 Je vous laisse transformer ce code pour retrouver la trame du projet précédent en partant du code du calendrier de base python.

1. Trouvez le template à définir (idem)
2. Transformation
3. réduction
4. CSS