Objectifs

Nous allons créer une liste de tâches !

Finalement, chaque tâche pourra être selectionnée, modifiée ou même supprimée. Vous devrez stoquer les données de manière persistante dans le "localstorage" de votre navigateur.

Start Projet

Nous allons le réaliser en pure JS. On dit en Vanilla JS.

Commençons par l'interface HTML de base. Créez un fichier index.html.

index.html

```
1. <h1>Tasks</h1>
2.  <form>
3.    <div>
4.      <label for="task">New Task</label>
```

```
5.      <input id="task" autocomplete="off" autofocus placeholder="Add New
    Task" type="text">
6.    </div>
7.    <button type="submit">Add Task</button>
8.  </form>
9.  <ol id="tasks">
10. </ol>
```

Ajoutez la bibliothèque bootstrap  index.html.

Il suffit d'ajouter un lien dans le header sur la bibliothèque.

Bootstrap fonctionne sur l'ajout de classes CSS sur les éléments.

Exemple de déclaration

La classe **class="mt-5"** sera un raccourci pour définir la marge en haut de taille 3 rem (48px). Voici les équivalences :

Where ***property*** is

- `m` - for classes that set `margin`

Where ***sides*** is

- `t` - for classes that set `margin-top` or `padding-top`

Where ***size*** is

- `5` - (by default) for classes that set the `margin` or `padding` to `$spacer * 3`

Ainsi le code:

HTML	résultat
<h1 **class="mt-5"**>Tasks</h1>	Tasks

index.html

```
1.  <h1 class="mt-5">Tasks</h1>
2.
3.  <form class="mb-3">
4.    <div class="input-group mb-3">
5.      <label class="input-group-text" for="task">New Task</label>
6.      <input id="task" class="form-control" autocomplete="off" autofocus
    placeholder="Add New Task" type="text">
7.    </div>
8.    <button type="submit" class="btn btn-primary">Add Task</button>
9.  </form>
10. <ol id="tasks" class="list-group">
11. </ol>
```

Ce document HTML met en place une application simple de liste de tâches avec Bootstrap pour le style et la mise en page.

Il comprend : un formulaire <form> pour ajouter de nouvelles tâches et une liste ordonnée pour afficher les tâches. Le formulaire et les éléments de saisie sont stylisés à l'aide des classes Bootstrap, et la fonctionnalité est censée être gérée par le module JavaScript inclus.

Créez un fichier todo.mj[1]s puis ajoutez le lien dans le fichier HTML.

```
<script type="module" src="./todo.mjs"></script>
```

Passons au code du module

todo.mjs

```
1. document.querySelector("form").onsubmit = () => {
2.   // 🪛Create new item for list
3.   const li = document.createElement("li");
4.   li.innerHTML = document.querySelector("#task").value;
5.
6.   // 🪛Add new item to task list
7.   document.querySelector("#tasks").append(li);
8.
9. };
```

🪛Testez le code ! Voir exemple :
https://codepen.io/SuperDupont/pen/NPKqMzB

Il semblerait que rien ne se passe !

[1] Nous reviendrons sur l'extension mjs (module)

😈Il y a un "bug" ?

🆘Le problème vient **du comportement de base** du formulaire.

Le comportement de base d'un formulaire est d'envoyer les données à un serveur et de rafraîchir la page.

Le comportement de base doit être inhibé pour pouvoir gérer le formulaire par notre javascript !

La principale différence entre un formulaire HTML et un document HTML habituel réside dans le fait que, généralement, les données collectées par le formulaire sont envoyées vers un serveur web.

Le problème ~~peut venir~~ vient du fait que le formulaire est soumis avant que le code JavaScript ne puisse s'exécuter.

🚀 Pour empêcher la soumission du formulaire, vous devez appeler event.preventDefault() dans le gestionnaire d'événement onsubmit ou simplement **return false**. Voici le code de todo.mjs mis à jour.

Le comportement de base est inhibé à l'aide de return false.

todo.mjs

```
1. document.querySelector("form").onsubmit = () => {
2.
3.   // Create new item for list
4.   const li = document.createElement("li");
```

```
    li.innerHTML = document.querySelector("#task").value;

    // Add new item to task list
    document.querySelector("#tasks").append(li);

    // 🔪Stop form from submitting
    return false;
};
```

🚀En action

🛠️ Au niveau de l'affichage, il reste encore une amélioration à réaliser. Nous remarquons en effet qu'une fois sauvée, la tâche reste dans le code de saisie.

Finalement, voici le code de base de todo.mjs

todo.mjs

```
document.querySelector("form").onsubmit = () => {
    // Create new item for list
    const li = document.createElement("li");
    li.innerHTML = document.querySelector("#task").value;

    // Add new item to task list
    document.querySelector("#tasks").append(li);

    // Clear input field
```

```
10. document.querySelector("#task").value = "";
11.
12. // Stop form from submitting
13. return false;
14.};
```

Lig. 10 : le formulaire est vidé de la valeur saisie.

addEventlistener

Nous allons adapter notre code todo.mjs pour utiliser la méthode "addEventlistener".

todo.mjs

```
1. document.querySelector("form").addEventListener("submit", (event) => {
2.   event.preventDefault();
3.
4.   // 🪛Create new item for list
5.   const li = document.createElement("li");
6.   li.innerHTML = document.querySelector("#task").value;
7.
8.   // 🪛Add new item to task list
9.   document.querySelector("#tasks").append(li);
10.
11. // 🪛Clear input field
12. document.querySelector("#task").value = "";
```

```
13.});
```

Lig. 1 : L'événement "submit" est déclenché lorsque l'utilisateur soumet le formulaire. Le code Javascript est exécuté avant que le formulaire ne soit soumis.

Lig. 2 : La méthode preventDefault(), rattachée à l'interface Event, indique à l'agent utilisateur que si l'évènement n'est pas explicitement géré, l'action par défaut ne devrait pas être exécutée comme elle l'est normalement. Plus simplement, cela empêche l'envoie par défaut du formulaire.

Création d'un template

Un template HTML est une méthode permettant de définir dans le code JS les structures HTML. C'est très utile dans le cadre de la programmation dynamique.

Nous allons mettre en place un **template**, pour indiquer clairement (en séparant les codes) ce que nous voulons ajouter à la liste à chaque fois que le formulaire est rempli. Nous utiliserons les backticks (`) qui incluent des expressions dynamiques. Le template permet donc de créer une structure HTML dynamique en fonction des données saisies par l'utilisateur. Idéaleme

Nous allons modifier le code de todo.mjs.

todo.mjs

```
1. // 🪛Define a template string for the list item
2. const template = `
3.   <li style="cursor: pointer;">{{data}}</li>
4. `;
5.
6. document.querySelector("form").addEventListener("submit", (event) => {
7.   event.preventDefault();
8.
9.   // Get the task value
10.  const taskValue = document.querySelector("#task").value;
11.
12.  // Create new list item by replacing the placeholder with the actual task
     value
13.  const liHTML = template.replace("{{data}}", taskValue);
14.
15.  // Add new item to task list
16.  document.querySelector("#tasks").insertAdjacentHTML('beforeend',
     liHTML);
17.
18.  // Clear input field
19.  document.querySelector("#task").value = "";
20.});
```

Lig. 1 : On sépare le code du template du reste du code.

Lig.3 : On écrit le code désiré ici <li style="cursor: pointer;">{{data}}. On utilise la convention d'écriture des templates les doubles accolades pour clairement indiquer quelle valeur doit être remplacée.[2]

[2] On pourrait utiliser d'autres codes que les doubles {{}}

Lig.13 : On remplace la valeur data entre les {{ }} par la valeur saisie !

Amélioration du comportement

Le comportement doit être corrigé. En effet, le bouton 

est valide même avec un formulaire vide !

Tasks

New Task Add New Task
Add Task

1.
2.
3.

Disabled

🛠️L'attribut booléen **disabled**, lorsqu'il est présent, rend l'élément non modifiable, non focalisable, ni même soumis avec le formulaire. L'utilisateur ne peut pas modifier le contrôle ou ses éléments descendants, ni leur donner la priorité.

Réécrivons le fichier todo.mjs

todo.mjs

```
// Define a template string for the list item
const template = `
  <li style="cursor: pointer;">{{data}}</li>
`;

const form = document.querySelector("form");
const taskInput = document.querySelector("#task");
const submitButton = form.querySelector("button[type='submit']");

// 🔪Disable the submit button initially
submitButton.disabled = true;

🔪// Add event listener to the input field to enable/disable the button
taskInput.addEventListener("input", () => {
  submitButton.disabled = taskInput.value.trim() === "";
});

form.addEventListener("submit", (event) => {
  event.preventDefault();

  // Get the task value
  const taskValue = taskInput.value;

  // Create new list item by replacing the placeholder with the actual task value
  const liHTML = template.replace("{{data}}", taskValue);

  // Add new item to task list
```

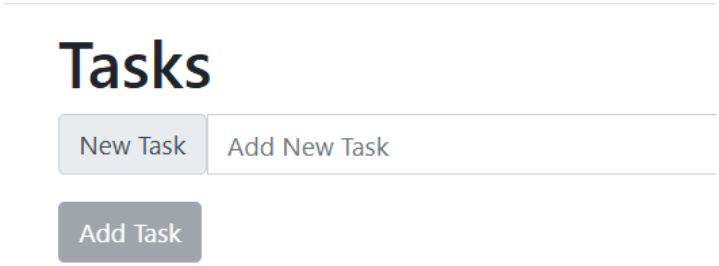
```
28.  document.querySelector("#tasks").insertAdjacentHTML('beforeend',
     liHTML);
29.
30.  // Clear input field and disable the button
31.  taskInput.value = "";
32.  submitButton.disabled = true;
33.});
```

Lig. 15 : Si la valeur est vide, on rend inactif le bouton. trim() permet de supprimer les tabulations ou espaces.

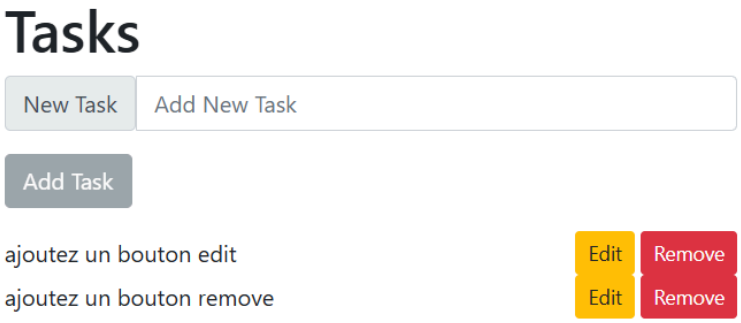
Lig. 30 : On améliore le comportement après l'ajout d'une tâche.

Améliorations du code !

Nous voulons obtenir le comportement suivant :



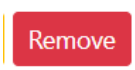
Ajoutez des gestions sur chaque tâche : 




```
1. // Define a template string for the list item
2. const template = `
3.   <li style="cursor: pointer;">{{data}}</li>
4. `;
5.
6. const form = document.querySelector("form");
7. const taskInput = document.querySelector("#task");
8. const submitButton = form.querySelector("button[type='submit']");
9.
10. // Disable the submit button initially
11. submitButton.disabled = true;
12.
13. // Add event listener to the input field to enable/disable the button
14. taskInput.addEventListener("input", () => {
15.   const isInputEmpty = taskInput.value.trim() === "";
16.   submitButton.disabled = isInputEmpty;
17.   submitButton.classList.toggle("btn-secondary", isInputEmpty);
18.   submitButton.classList.toggle("btn-primary", !isInputEmpty);
19. });
```

```
20.
21. form.addEventListener("submit", (event) => {
22.   event.preventDefault();
23.
24.   // Get the task value
25.   const taskValue = taskInput.value;
26.
27.   // Create new list item by replacing the placeholder with the actual task
      value
28.   const liHTML = template.replace("{{data}}", taskValue);
29.
30.   // Add new item to task list
31.   document.querySelector("#tasks").insertAdjacentHTML("beforeend",
      liHTML);
32.
33.   // Clear input field and reset the button
34.   taskInput.value = "";
35.   submitButton.disabled = true;
36.   submitButton.classList.toggle("btn-secondary", true);
37.   submitButton.classList.toggle("btn-primary", false);
38. });
```

Modifions le template

Il nous faut ajouter un bouton 

Pré-requis :

1. classList.contains
2. event.target
3. node.parent
4. node.remove

```
// 🪛Define a template string for the list item with a remove button
const template = `
  <li class="d-flex justify-content-between align-items-center" style="cursor: pointer;">
    <span>{{data}}</span>
    <button class="btn btn-danger btn-sm remove-btn">Remove</button>
  </li>
`;

const form = document.querySelector("form");
const taskInput = document.querySelector("#task");
const submitButton = form.querySelector("button[type='submit']");

// Disable the submit button initially
submitButton.disabled = true;

// 🪛Add event listener to the input field to enable/disable the button
taskInput.addEventListener("input", () => {
  const isInputEmpty = taskInput.value.trim() === "";
  submitButton.disabled = isInputEmpty;
  submitButton.classList.toggle("btn-secondary", isInputEmpty);
  submitButton.classList.toggle("btn-primary", !isInputEmpty);
```

```
});

form.addEventListener("submit", (event) => {
  event.preventDefault();

  // Get the task value
  const taskValue = taskInput.value;

  // Create new list item by replacing the placeholder with the actual task value
  const liHTML = template.replace("{{data}}", taskValue);

  // Add new item to task list
  document.querySelector("#tasks").insertAdjacentHTML("beforeend", liHTML);

  // Clear input field and reset the button
  taskInput.value = "";
  submitButton.disabled = true;
  submitButton.classList.toggle("btn-secondary", true);
  submitButton.classList.toggle("btn-primary", false);
});

// 🚀Add event listener to the task list to handle remove button clicks
document.querySelector("#tasks").addEventListener("click", (event) => {
  if (event.target.classList.contains("remove-btn")) {
    event.target.parentElement.remove();
  }
```

52.});

Travail personnel

Et maintenant l'édition avec un bouton 

avec la possibilité de faire apparaître un formulaire de saisie

Et nous finirons par examiner comment utiliser un enregistrement local comme conserver les tâches.

Voici un lien vers le git : https://github.com/dupontdenis/TODO-TD.git

Annexe

Création d'un template Element

1. // Create a template for the list item
2. const template = **document.createElement("template");**
3. template.innerHTML = `

```
    <style>
      li {
        cursor: pointer;
      }
    </style>
    <li></li>
`;

document.querySelector("form").addEventListener("submit", (event) => {
  event.preventDefault();

  // Clone the template content
  const clone = document.importNode(template.content, true);
  const li = clone.querySelector("li");
  li.textContent = document.querySelector("#task").value;

  // Add new item to task list
  document.querySelector("#tasks").appendChild(li);

  // Clear input field
  document.querySelector("#task").value = "";
});
```

Learn how to !

Créer une liste de tâches peut être réalisé dans avec d'autres technologies. voici comment on peut la réaliser en React 🚀lien. On peut évidemment l'associer à un serveur comme Django …

Ou, utiliser l'orienté-objet et le modèle MVC

🧙‍♂️https://dupontdenis.github.io/TODOes6/